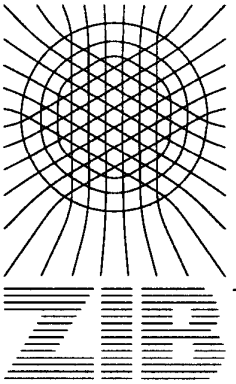
Konrad-Zuse-Zentrum für Informationstechnik Berlin



CRAY–Handbuch

Einführung in die Benutzung der CRAY X–MP unter UNICOS 5.1

Technical Report TR 90–3

Dieses Handbuch ist eine vollständig überarbeitete Neuauflage unserer Schrift

CRAY-Handbuch
Einführung in die Benutzung der CRAY X-MP unter UNICOS
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Technical Report TR 89-1 (Januar 1989)

Insbesondere waren Hubert Busch, Hans-Christian Hege, Wolfgang Stech und Wilhelm Vortisch an der Überarbeitung dieser Schrift beteiligt; die redaktionelle Arbeit lag bei Wolfgang Stech.

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Straße 10
1000 Berlin 31
Verantwortlich: Dr. Klaus André
Umschlagsatz und Druck: Rabe KG Buch- und Offsetdruck Berlin

ISSN 0933-789X

VORWORT ZUR AUFLAGE JUNI 1990

Um nach dem Wechsel des Betriebssystems UNICOS am 27. März 1990 von Version 4 auf Version 5.1 den Benutzern wieder ein aktuelles Handbuch anzubieten, haben wir in dieser Auflage die Änderungen, die sich durch die neue Betriebssystem-Version ergeben haben, eingearbeitet. Den Zugang zur CRAY über TCP/IP-Protokolle bzw. das Wissenschaftsnetz (WiN) haben wir durch Einfügen eines neuen Kapitels 6 berücksichtigt (was eine Verschiebung der alten Kapitelnummern ab 6 um eins nach hinten zur Folge hatte). Das Kapitel 12 "Einführung in die Optimierung der Rechenzeit" wurde völlig überarbeitet und erheblich erweitert, so daß man sich nun über alle aktuellen Methoden, seine Programme für unsere CRAY zu optimieren, umfassend und ausführlich informieren kann. Im übrigen wurden alle Kapitel ergänzt, gründlich überarbeitet und der Anhang um eine separate Literaturliste erweitert.

Zwar sind alle bisher bekannt gewordenen Fehler korrigiert worden; trotzdem bitten wir um regen Gebrauch der im Anhang E beigefügten Formulare, um uns nicht nur Fehler, sondern insbesondere Ergänzungs- oder Änderungswünsche mitzuteilen.

Berlin, im Juni 1990 Wolfgang Stech

1. Überblick über die CRAY

1.1 Allgemeine Informationen über die CRAY X-MP des ZIB

1.1.1 Rechenarten

Die CRAY X-MP ist ein Vektorrechner. Ihr Befehlsvorrat umfaßt neben den üblichen arithmetischen Befehlen für einzelne Zahlen auch Vektorbefehle, die gleichartige Operationen auf Zahlenkolonnen (Vektoren) ausführen. Jeder dieser Vektorbefehle stößt eine größere Anzahl von Operationen an, die stark überlappt ablaufen. Die Überlappung führt zu einer hohen Zahl von Operationen je Zeiteinheit und damit zu der hohen Rechenleistung der Anlage.

Die Überlappung von Operationen wird auch in konventionellen Rechnern zur Leistungssteigerung eingesetzt. Im Vektorrechner ist der Strom der Operanden und die Steuerung des Rechenvorgangs jedoch in besonderem Maße aufeinander abgestimmt. Die Zusammenfassung gleichartiger Operationen zu mächtigen Vektorbefehlen ist das besondere Merkmal der Vektorrechner.

Die folgende Schleife wird auf einem konventionellen Rechner durch wiederholte Ausführung einer Folge arithmetischer Befehle abgearbeitet:

```
      DO 100 I=1,50
100   A(I) = B(I)*C(I)
```

Auf der CRAY werden die Multiplikationen mit einem einzigen Befehl angestoßen und laufen dann überlappt ab. Beispielsweise beginnt das Rechenwerk mit der Berechnung von B(2)*C(2) kurz nachdem die Berechnung von *B(1)*C(1)* angelaufen ist, und lange ehe der Wert des Produkts *B(1)*C(1)* ermittelt ist.

Vektorielle Verarbeitung

Die an einer Vektoroperation beteiligten Datenströme bezeichnet man als Vektoren. Ein Vektor im Sinne des Rechners CRAY X-MP ist eine Folge von Werten, die im Speicher aufeinanderfolgend oder in gleichem Abstand gespeichert sind. In FORTRAN sind dies z.B. eindimensionale Felder, Spalten, Zeilen und Diagonalen einer Matrix oder systematische Ausschnitte solcher Einheiten. Vektoren können durch die üblichen arithmetischen Operationen verknüpft werden.

Die hohe Leistung des Vektorrechners kann besonders dann ausgeschöpft werden, wenn die Vektoren lang sind. Für Vektoren mit weniger als fünf Elementen bringen die Vektorbefehle gegenüber wiederholter Ausführung konventioneller Befehle noch keinen Gewinn, da der Prozessor eine gewisse Vorlaufzeit benötigt, ehe er die volle Vektorleistung erreicht. Bei Vektoren der Länge 30 ist der Gewinn erheblich und schon bei der Länge 64 für die CRAY optimal.

Die Vektorlänge 64 spielt für die CRAY deshalb eine zentrale Rolle, weil die Daten für die Verarbeitung in speziellen Vektorregistern zurechtgelegt werden, die je 64 Elemente aufnehmen können. Der Anwender braucht jedoch nicht eine Aufteilung der Vektoren in vierundsechziger Gruppen vorzunehmen: dies erledigt der FORTRAN-Compiler automatisch. Das Augenmerk ist lediglich darauf zu richten, möglichst lange Vektoren zu bearbeiten.

Arithmetische Ausdrücke

Arithmetische Ausdrücke mit Vektoren und Skalaren als Operanden werden dann besonders schnell ausgeführt, wenn möglichst viele Zwischenergebnisse in Registern gehalten werden können, so daß die Anzahl der zeitaufwendigen Zugriffe zum Hauptspeicher gering ist. Dies erreicht man, indem jeweils möglichst viele Operanden zu einem arithmetischen Ausdruck zusammengefaßt werden.

Bei einigen Kombinationen von arithmetischen Verknüpfungen können mehrere Vektorbefehle vollständig gleichzeitig ablaufen, wodurch lokal eine Verdopplung der Leistung erreicht wird, z.B.

```
A(I) = C(I) + X * D(I)
```

Dies ist immer dann möglich, wenn verschiedene Prozessoreinheiten angesprochen werden, z.B. die Multiplikationseinheit und die Additionseinheit. Dabei brauchen die Operanden nicht voneinander unabhängig zu sein, da der Rechner mit Hilfe einer als *chaining* bezeichneten Technik die Resultate einer Operation, z.B. der Multiplikation, als Operanden der nächsten Operation, z.B. der Addition, unmittelbar verwendet.

Theoretisch ist jede CPU (Central Processing Unit) der CRAY X-MP in der Lage, bis zu 210 Millionen Gleitkommaoperationen in der Sekunde auszuführen. Dies kann allerdings nur im Idealfall erreicht werden, in dem drei verschiedene Operationen (z.B. Betragsbildung, Addition und Multiplikation) durch Verkettung (*chaining*) verknüpft werden können. Dem Grenzwert wird man sich jedoch in der Praxis nur in seltenen Fällen und bei sehr speziellen Fragestellungen nähern können. Die maximale Rechenleistung kann auch nur dann erreicht werden, wenn bei der Rechnung nicht zu viele Hauptspeicherzugriffe notwendig sind und wenn die Parallelverarbeitung fortlaufend ausgenutzt werden kann. In der Praxis ist ein Mehrfaches von 30 Millionen Gleitkommaoperationen pro Sekunde als mittlere Leistung über einen Programmlauf erreichbar.

Die Höchstleistung bei der Vektorverarbeitung wird stark beeinträchtigt, wenn der Datenstrom für die Verknüpfungen nicht ungehindert fließen kann. Dies ist zum Beispiel der Fall, wenn die Operationen von Bedingungen abhängig sind:

```
      DO 100 I=1,3000
      IF (X(I) .GT. 0.0) Y(I)=X(I)*Z
100   CONTINUE
```

oder wenn Resultate als Operanden der gleichen Operation benötigt werden

```
      DO 200 I=2,3000
200   X(I) = X(I-1)*Y(I)
```

In dem letzten Beispiel wird in der Anweisung 200 z.B. das Element X(2) für die Berechnung von X(3) benötigt. Die zweite Berechnung kann also nicht gleichzeitig mit der ersten ablaufen. Der Compiler erkennt derartige Probleme und generiert hier konventionelle Befehle für Einzelbearbeitung. Wenn aus einem dieser oder anderen Gründen Vektorverarbeitung nicht möglich ist, informiert der Compiler über die Ursache in den Übersetzungslisten. Ausführliche Hinweise über die Möglichkeiten der Autovektorisierung des FORTRAN-Übersetzers und der Einflußnahme durch den Benutzer finden sich im Kapitel 11.

Neben den Techniken zur Optimierung eines Programmes auf einer CPU gewinnen neuerdings Techniken an Bedeutung, die die Leistungen mehrerer oder aller Prozessoren eines DV-Systems auf ein einziges Programm zu konzentrieren (siehe auch Kapitel 12). Auf der CRAY stehen hierfür die Techniken des *multitasking*, *macrotasking*, *microtasking* und *autotasking* zur Verfügung. Da die Anlage des ZIB nur über zwei Prozessoren verfügt, werden nur wenige existierende Anwendungen auf der Anlage des ZIB aus diesen Techniken Gewinn ziehen können. Sie sind daher in dieser Schrift nicht beschrieben. Falls Sie z.B. für künftige Programmentwicklungen Näheres über diese Techniken wissen wollen, wenden Sie sich bitte an Ihr Rechenzentrum oder direkt an das ZIB.

1.1.2 Speicherhierarchie

Die Notwendigkeit, den Prozessor mit einem breiten, ungehinderten Datenstrom zu versorgen, spiegelt sich in der feinen Abstufung der Speicherhierarchie und der Transportwege der CRAY X-MP wider. Die Kenntnis der Speicherhierarchie setzt den Anwender in vielen Fällen überhaupt erst in die Lage, seine Jobs so zu

konfigurieren, daß sie die volle Leistung der CRAY X-MP/24 ausnutzen.

Register

Die in dem umfangreichen Registersatz vorliegenden Daten können verzögerungsfrei und simultan vom Prozessor erreicht werden.

Hauptspeicher

Der Hauptspeicher der CRAY X-MP/24 (4 Million Worte zu je 64 Bit) ist durch Überlappung und andere technische Maßnahmen so ausgelegt, daß nach einer gewissen Anlaufzeit pro CPU zweimal 105 Millionen Worte pro Sekunde aus dem Hauptspeicher in Vektorregister und einmal 105 Millionen Worte pro Sekunde vom Vektorregister in den Hauptspeicher geladen werden können.

Erweiterungsspeicher SSD (Solid-State Storage Device)

Die schnellste Möglichkeit, Daten aus dem relativ kleinen Hauptspeicher auszulagern, bietet der Erweiterungsspeicher SSD. Er hat im ZIB eine Kapazität von 128 Millionen Worten (1 GByte) und ist mit der Zentraleinheit über einen Hochgeschwindigkeitskanal (ca. 1 GByte/s) verbunden. Der SSD wird für vielfältige Aufgaben (u. a. auch als Stufe in der Speicherhierarchie) vom Betriebssystem eingesetzt, ist aber auch direkt vom Benutzerprogramm aus ansprechbar.

Ein-/Ausgabesystem

Durch einen separaten, der CPU vorgeschalteten Ein-/Ausgaberechner wird die Zentraleinheit von allen E/A-Vorgängen entlastet. Das Ein-/Ausgabesystem bedient insbesondere die Plattenspeicher und die Vorrechner und puffert Daten in erheblichem Umfang, damit die Zentraleinheit möglichst selten durch Transporte unterbrochen und behindert wird. Das Ein-/Ausgabesystem ist durch einen schnellen Kanal mit der Zentraleinheit verbunden (Transferrate 100 MByte/s).

Plattenspeicher

An den Rechner CRAY X-MP sind schnelle Plattenspeicher angeschlossen. Um auch hier eine hohe Transportrate zu erreichen, kann die Transportleistung aller Plattenspeicher simultan erbracht werden. Damit ist es möglich, durch Verteilung der zu einem Programmlauf gehörenden Dateien auf verschiedene Platten auch individuell Transportleistungen bis zu 50 MByte/s zu erhalten. Einige auf der UNIX-Workstation *serv01* installierte Plattenbereiche sind über *NFS* (Network Filesystem) auch der CRAY zugeordnet, so daß sie direkt von der CRAY aus über normale Befehle wie *cd*, *cp* u.ä. angesprochen werden können; die tatsächliche Transferrate liegt jedoch unter 100 KByte/s.

Vorrechner

Die letzte Stufe in der Speicherhierarchie bilden die Vorrechner mit den dort angeschlossenen Speichern. Bei den Vorrechnern handelt es sich um handelsübliche Universalrechner. Sie haben die Aufgabe, den Vektorrechner mit Aufträgen zu versorgen, Resultate entgegenzunehmen und weiterzuleiten sowie Daten zu speichern.

1.1.3 Verbindung mit dem Benutzer

Stations

Zum Liefer- und Verantwortungsumfang der Fa. CRAY gehört in jedem Vorrechner ein "Station" genanntes Teilsystem, welches die Schnittstelle zwischen Vektorrechner und Benutzer darstellt. Die Station nimmt Batchjobs entgegen, liefert Resultate (Listen, Mitteilungen usw.) ab, überträgt Dateien zwischen Vektorrechner und Vorrechner, liefert Statusinformationen über die Bearbeitung der Jobs an der CRAY und gibt Möglichkeiten der Beeinflussung eigener Jobs. Die Stationsoftware ist ihrem jeweiligen "Wirtssystem" angepaßt: unter NOS/BE z.B. kommuniziert sie unmittelbar mit den Ein-/Ausgabe-Queues des Betriebssystems, so daß der Rechner CRAY X-MP, abgesehen von der abweichenden Steuersprache, als Mitglied eines homogenen Rechnernetzes angesprochen wird.

Programmiertechnik

Die Einfachheit des Zugangs darf nicht darüber hinwegtäuschen, daß Programme, die auf dem Vektorrechner zeitgünstig laufen sollen, zweckmäßig gestaltet sein müssen. Insbesondere ist die Höchstleistung der CRAY X-MP nur mit speziellen Programmen erreichbar. Zwar ist die Maschine auch im nicht-vektoriellen Bereich sehr leistungsfähig, jedoch stellen sich im Vergleich zu Universalrechnern erhebliche Verkürzungen der Programmlaufzeiten erst ein, wenn die Vektorverarbeitung im Programm überwiegt und wenn sie für den Vektorrechner geeignet formuliert ist. Unterstützung für die Formulierung geben spezielle Bibliotheken und Erweiterungen der Programmiersprachen FORTRAN, C und PASCAL sowie die Literatur des Herstellers.

1.2 Norddeutscher Vektorrechnerverbund

Mit den Berliner Universitäten, der Bundesanstalt für Materialforschung und -prüfung (BAM), dem Hahn-Meitner-Institut Berlin GmbH (HMI) und den Ländern Schleswig-Holstein und Niedersachsen bestehen Kooperationsvereinbarungen über die Benutzung des Vektorrechners im ZIB einschließlich seiner Vorrechner. Im Rahmen dieser Kooperationsvereinbarungen wurde festgelegt, daß für die Zulassung und Betreuung von Benutzern aus den Bereichen dieser Kooperationspartner jeweils diese Partner zuständig sind. Den Berliner Benutzern stehen damit im Rahmen des Norddeutschen Verktorrechnerverbundes auch die Landesvektorrechner CRAY X-MP/216 der Universität Kiel und die Siemens VP 200 der Universität Hannover zur Verfügung. Ihre Benutzung kommt insbesondere dann in Betracht, wenn besonders hohe Anforderungen an die Rechnerkapazität (Hauptspeicher) vorliegen. Das ZIB selbst läßt darüberhinaus Benutzer aus Wissenschaft und Industrie zu, wenn dies mit seinen eigenen Forschungsprojekten vereinbar ist. Die jeweilige Rechenerlaubnis erteilen im Rahmen der oben genannten Kooperationsvereinbarungen folgende Stellen (siehe Kapitel 1.7):

für Berlin:

Zentraleinrichtung für Datenverarbeitung der Freien Universität (FUB),
Zentraleinrichtung Rechenzentrum der Technischen Universität (TUB),
Rechenzentrum der Bundesanstalt für Materialforschung und -prüfung (BAM),
Bereich Datenverarbeitung/Mathematik des Hahn-Meitner-Instituts Berlin GmbH (HMI),

für Niedersachsen:

Regionales Rechenzentrum für Niedersachsen der Universität Hannover,
Rechenzentrum der Technischen Universität Braunschweig,
Gesellschaft für wissenschaftliche Datenverarbeitung GmbH Göttingen,
Rechenzentrum der Universität Clausthal,
Rechenzentrum der Universität Osnabrück,

für Schleswig-Holstein:

Rechenzentrum der Universität Kiel,

für Projekte mit dem ZIB:
Konrad-Zuse-Zentrum für Informationstechnik, Berlin.

Jeder interessierte Benutzer kann bei einem für ihn zuständigen Rechenzentrum auf Antrag eine Erlaubnis zur Zulassung zum Rechner CRAY X-MP sowie für einen oder mehrere Vorrechner erhalten. Bei Zulassung erhält der Benutzer einen Abrechnungsnamen (userid) und ein Kennwort (password) für die CRAY und den jeweiligen Vorrechner. Antragsformulare stehen in den Rechenzentren zur Verfügung.

Abrechnungsnamen und Kennworte sollten zum Schutz vor Mißbrauch geheimgehalten werden. Festgestellter Mißbrauch ist dem eigenen Rechenzentrum und dem ZIB umgehend mitzuteilen. Das Ausscheiden eines Benutzers aus dem Institut ist dem Rechenzentrum anzuzeigen; ohne Genehmigung der Rechenzentren sind Benutzernummern nicht auf andere Personen übertragbar. Die Nutzungsberechtigung gilt in der Regel für ein Kalenderjahr, kann aber jeweils um ein weiteres Jahr verlängert werden.

1.3 Konfiguration

1.3.1 CRAY X-MP

Die Rechenanlage CRAY X-MP ist wie folgt konfiguriert:

Zentraleinheit

Die Zentraleinheit CRAY X-MP verfügt über zwei Prozessoren, die mit jeweils 13 unabhängigen und segmentierten Funktionseinheiten ausgerüstet sind:

zwei für Adressrechnung:
* Addition (24-Bit)
* Multiplikation(24-Bit)

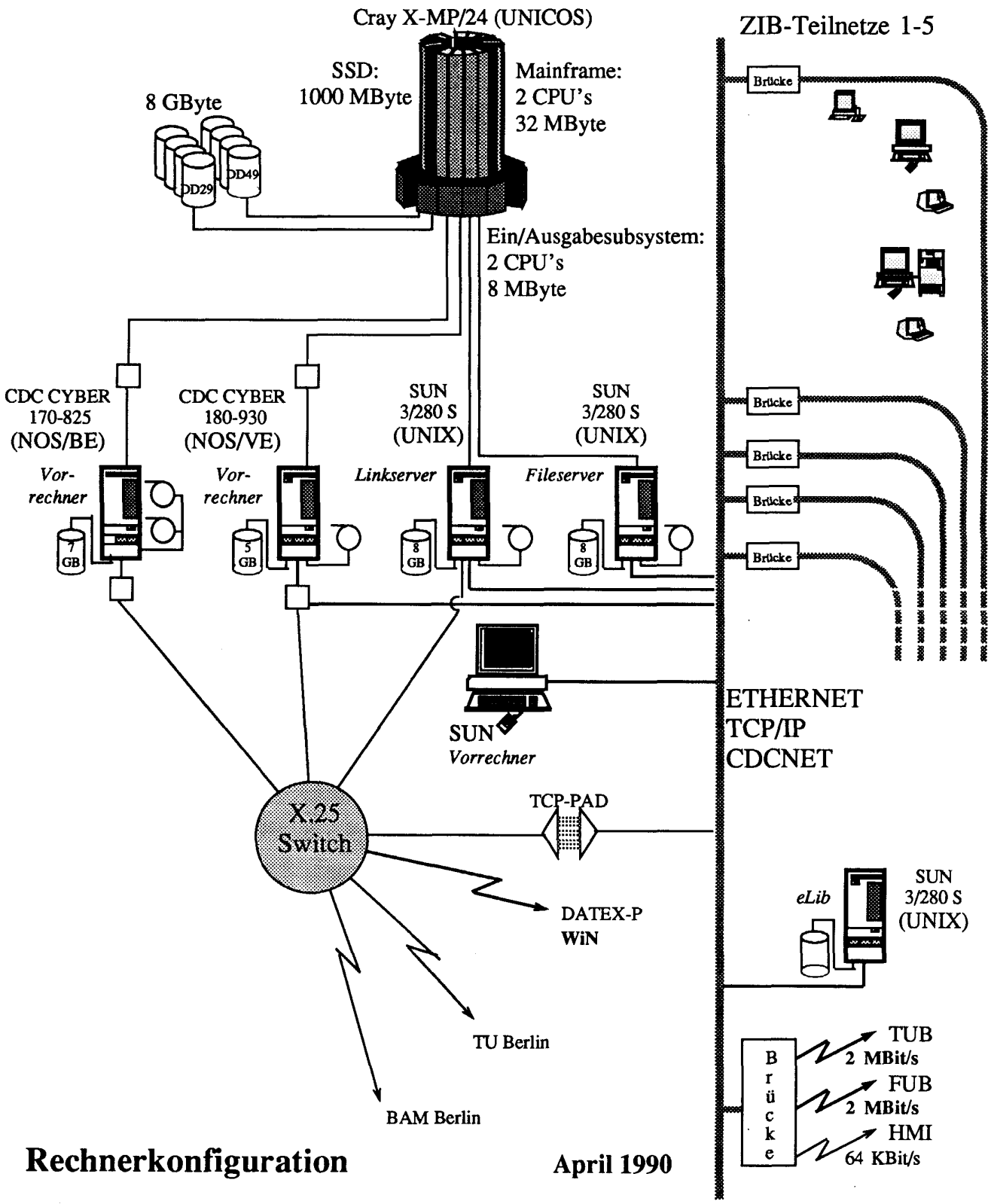
vier für skalare Grundoperationen:
* Addition (64-Bit)
* Shift
* Logische Operationen
* Population Count (Anzahl der auf 1 oder 0 gesetzten Bits zählen)

vier für vektorielle Grundoperationen:
* Addition (64-Bit)
* Shift
* Logische Operationen
* Population Count (Anzahl der auf 1 oder 0 gesetzten Bits zählen)

drei sowohl für skalare als auch vektorielle Gleitkommaoperationen:
* Addition
* Multiplikation
* Reziproke Approximation (statt echter Division)

Es gibt fünf Arten von schnellen Registern:

* 8 Vektorregister (V) mit je 64 Worten zu 64 Bit
* 8 Skalarregister (S) zu 64 Bit
* 64 Skalarregister (T) zu 64 Bit
* 8 Adressregister (A) zu 24 Bit
* 64 Adressregister (B) zu 24 Bit.



Rechnerkonfiguration **April 1990**

Die Funktionseinheiten holen die Operanden und speichern die Ergebnisse nur aus den bzw. in die A-, S- und V-Register. Die B- und T-Register werden als schneller Zwischenspeicher für skalare Operationen verwendet.

Die interne Zykluszeit der CRAY X-MP beträgt 9.5 ns.

Die Hardware ist in ECL-bipolar-Technik realisiert; der Hauptspeicher besteht aus 4 Millionen direkt adressierbarer Worte zu je 64 Bit. Jede CPU verfügt über drei unabhängige Zugriffspfade zum Hauptspeicher; zwei dienen zum Laden von Vektoren, der dritte zum Speichern von Vektoren und zum Laden und Speichern von Skalaren. Der Speicher besteht aus 32 Bänken, die so verschränkt sind, daß adreßmäßig aufeinander folgende Worte in unterschiedlichen Bänken liegen. Jede Bank kann alle 38 ns (4 Zyklen) erneut angesprochen werden. Auf Grund der Verschränkung können daher Elemente von Vektoren in der Regel alle 9,5 ns (der Zykluszeit) geladen oder gespeichert werden.

Der SSD ist ausschließlich über den Hochgeschwindigkeitskanal (1 GByte/s) mit der Zentraleinheit verbunden. Er hat eine Kapazität von 128 MW (1 GByte) und ist in ECL-Technik realisiert. Wegen der Startup-Zeiten des Kanals ist es empfehlenswert, jeweils größere Blöcke von Daten zu transportieren. Der Anwender kann dies durch entsprechende Blockstruktur der Dateien erreichen, die er auf dem SSD anlegt (im ZIB ausschließlich als Scratch-Files).

Ein-/Ausgabesystem und Plattenspeicher

Die CRAY X-MP im ZIB verfügt über ein Ein-/Ausgabesystem, bestehend aus 2 Prozessoren (ein *Master* Ein-/Ausgabe-Prozessor, ein Puffer Ein-/Ausgabe-Prozessor) sowie eine Million Worte Pufferspeicher je 64 Bit. Das Ein-/Ausgabesystem dient insbesondere der Steuerung der Plattenlaufwerke und der Kommunikation mit den Vorrechnern.

Als Hintergrundspeicher dienen vier Plattenlaufwerke DD-29, Speicherkapazität je 600 MByte, Transferrate 4,5 MByte/s sowie vier Plattenlaufwerke DD-49, Speicherkapazität je 1,2 GByte, Transferrate 11 MByte/s.

Die Anlage CRAY X-MP verfügt (abgesehen von einem angeschlossenen Wartungsrechner mit Drucker und Magnetbandstation) nicht über Peripheriegeräte wie Magnetbandgeräte, Drucker, Kartenleser oder Sichtgeräte.

Die Verbindung mit der Außenwelt geschieht ausschließlich über Kanalverbindungen zwischen dem Ein-/Ausgabesystem und den Vorrechnern CD CYBER 180-930 (NOS/VE-System), CD CYBER 170-825 (NOS/BE-System) und SUN 3/280 (UNIX-System). Die theoretische Transferrate der Kanäle beträgt CRAY-seitig je 6 MByte/s.

1.3.2 Der Vorrechner CD CYBER 180-930

* Zentraleinheit (CPU) vom Typ 930,
  10 periphere I/O-Prozessoren (PPU's)
  32 KByte Cache Speicher
  32 MByte Hauptspeicher (SECDED)
  80 MByte/s Übertragungsrate zum Hauptspeicher
  120 ms Zugriffszeit
  8800 GByte virtueller Adressraum
  12 Kanäle mit 10 bzw. 20 MByte/s Übertragungsrate mit direktem Hauptspeicherzugriff (DMA)

* 1 Platten-Speichereinheit CD 9853-G4 mit 4 Laufwerken
  Gesamtkapazität 4,6 GByte
  mittlere Zugriffszeit 16 ms
  Übertragungsrate 3 MByte/s

* 1 Magnetbandgerät CD 639-1 mit zugehörigem
  Steuerwerk CD 7221-11, 9-Spur, 1600/6250 bpi,
  Übertragungsgeschw. 25 inch/s (156 KByte/s)

* integrierte Ethernet-Schnittstelle mit Anschluß an das Ethernet des ZIB, welches über Leitungen mit 2 Mb/s verbunden ist mit dem jeweiligen Ethernet der Freien Universität Berlin und der Technischen Universität Berlin.

1.3.3 Der Vorrechner CD CYBER 170-825

* Zentraleinheit (CPU) vom Typ 825,
  20 periphere Processoren (PPU's),
  2 MW Hauptspeicher je 60 Bit (MOS-Technik)
  24 Kanäle mit Übertragungsraten bis 4 MByte/s

* 6 Festplattendoppellaufwerke CD 885-12 mit 2 Steuereinheiten CD 7155-11
  Speicherkapazität je Doppellaufwerk 1,4 GByte,
  Übertragungsrate 1,6 MByte/s

* 2 Magnetbandstationen CD 679-7 (9-Spur) mit einer Schreibdichte von 1600 bpi bzw. 6250 bpi, einschließlich einer Steuereinheit CD 7021-31 mit einer Übertragungsrate von 1,25 MByte/s (bei 6250 bpi)

* 1 Zeilendrucker

* 2 Rechner ATM CLASSIC 7830 zum Anschluß der CYBER 825 an das Deutsche Forschungsnetz (DFN).

1.3.4 Die UNIX-Vorrechner

Die CRAY ist über zwei Front-End-Interface-Geräte des Typs FEI 3 mit zwei UNIX-Rechnern SUN 3/280 S (*serv01* und *serv02*) verbunden. Diese Rechner dienen als Dateiserver für sämtliche Workstations des ZIB.

* Zentraleinheit (CPU) vom Typ MC68020,
  8 MByte Hauptspeicher
  3 Kanäle mit Übertragungsraten bis 3 MByte/s
  1 Front-End-Interface FEI-3 S
  1 Magnetbandstation cipher M990 (9-Spur, (1/2")) mit einer Schreibdichte von 1600, 3200 oder 6250 bpi

  serv01:
* 6 Festplattenlaufwerke Fujitsu M2382K; Speicherkapazität je 800 MByte,
  Übertragungsrate 3 MByte/s,
  sowie (für das System) 2 Festplattenlaufwerke Hitachi DK815-10
  1 Streamer (1/4"-Magnetbandkasettengerät)
  64 KByte X.25-Anschluß über MCP-Interface

  serv02:
* 8 Festplattenlaufwerke Fujitsu M2382K; Speicherkapazität je 800 MByte,
  Übertragungsrate 3 MByte/s,
  sowie (für das System) 2 Festplattenlaufwerke Hitachi DK815-10

Zusätzlich steht ein weiterer UNIX-Rechner *ufer* (UNIX front end relay), derzeit eine SUN 3/60, für alle CRAY-Benutzer zur Verfügung. Es ist vorgesehen, diesen Rechner gemeinsam mit der CRAY als eine *yellow*

page domain zu betreiben: das bedeutet, daß jeder auf der CRAY zugelassene Benutzer auch zugelassener Benutzer auf dieser SUN ist und daß das Paßwort auf dieser SUN identisch ist mit dem auf der CRAY. *ufer* hat als Rechner keine eigenständige Bedeutung, insbesondere dürfen hier keine eigenständigen Aufgaben bearbeitet werden; er dient (wie die CD-Vorrechner) nur zur Vor- oder Nachbereitung von CRAY-Jobs. Insbesondere können Dateien, die auf den oben aufgeführten Plattenlaufwerken liegen, auch direkt auf diesem Rechner angesprochen, editiert oder mit FTP (siehe Kapitel 6.2) empfangen oder verschickt werden.

1.4 Einbindung der CRAY in Rechnernetze

Über EARN ist die CRAY X-MP des ZIB seit 26.3.1989 nicht mehr erreichbar.

1.4.1 Deutsches Wissenschaftsnetz WiN

Zur Datenkommunikation mit Übertragungsprotokollen auf der Basis der international anerkannten Empfehlung X.25 mit dem In- und Ausland bietet die Deutsche Bundespost seit mehreren Jahren den DATEX-P-Dienst an. Ähnlich wie bei der Sprachübertragung zwischen zwei menschlichen Kommunikationspartnern werden innerhalb des DATEX-P-Netzes auf gewählten Verbindungen Daten zwischen DV-Anlagen ausgetauscht. Anders als beim Telefonieren ermitteln sich die Gebühren für ein solches Gespräch aber nicht aus der geographischen Entfernung der Gesprächspartner und der zeitlichen Dauer des Gesprächs, sondern zu einem wesentlichen Teil aus dem Datenvolumen sowie der Verbindungszeit. Hinzu kommt noch ein geringer Zuschlag für die Herstellung der Verbindung.

Diese nicht im voraus kalkulierbaren Kommunikationskosten können in öffentlichen Haushalten (und um solche handelt es sich in der Regel im Wissenschaftsbereich) nicht etatisiert werden. Daher wurden in einigen Bundesländern als Zwischenlösung Rechnernetze zur Datenkommunikation auf X.25-Basis auf der Grundlage von Direktrufverbindungen ("Standleitungen") errichtet. In Berlin entstand ein Netz, an das neben den beiden (West-) Berliner Universitäten einige Institute aus dem Wissenschaftsbereich und auch die Universitäten in Hannover und Braunschweig sowie die GWD in Göttingen angeschlossen waren.

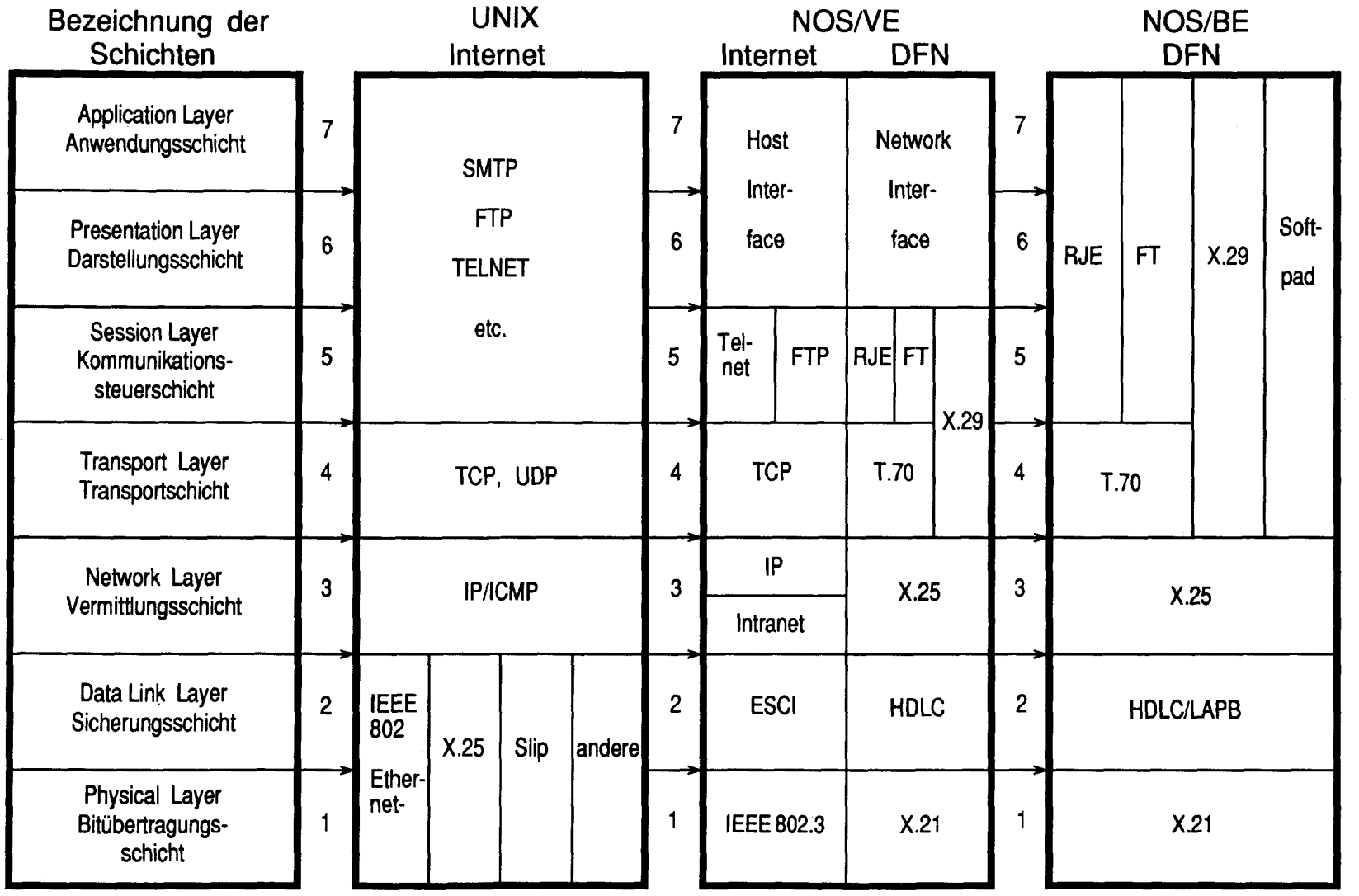
Parallel zu diesen Entwicklungen wurde zwischen dem DFN-Verein und der Deutschen Bundespost Telekom (DBP-T) ein Vertrag über ein kostengünstiges und leistungsfähiges Datenübertragungsnetz speziell für die wissenschaftliche Kommunikation auf der Basis von X.25, das sog. X.25-Wissenschaftsnetz (WiN), geschlossen.

Die besonderen Merkmale des WiN sind u. a.:

* feste monatliche Anschlußgebühren,
* keine Volumen- und Zeitgebühr, daher kostenlose Kommunikation innerhalb des WiN,
* gleiche Dienstgüte und Zuverlässigkeit wie DATEX-P,
* die Betriebsverantwortung liegt bei der DBP-T,
* freier und derzeit unentgeltlicher Zugang von und in ausländische Wissenschaftsnetze,
* freier Zugang zum öffentlichen DATEX-P-Netz und zu internationalen öffentlichen Netzen - jedoch gegen Zahlung der nutzungsabhängigen Gebühren der DBP,
* Reduzierung der Gebühren für Zubringerleitungen um bis zu 50%.

Über das WiN können grundsätzlich alle auf der genormten Schnittstelle X.25 aufbauenden Kommunikationsprotokolle verwendet werden. Insbesondere gehören hierzu die Kommunikationsdienste des DFN-Vereins (Dialog gemäß X.3, X.28, X.29, *e-mail* gemäß X.400, FTAM, DFN-FT und DFN-RJE) und die auf TCP/IP basierenden Protokolle (TELNET, FTP und SMTP (mail)).

ISO - Referenz-Modell



Das WiN umfaßt das Gebiet der Bundesrepublik Deutschland sowie das Land Berlin und ist fernmelderechtlich ein Bestandteil des öffentlichen Telekommunikationsnetzes. Es darf nicht für Sprachkommunikation genutzt werden. Anwender des WiN können nur Mitglieder des DFN-Vereins werden, also der Kreis der öffentlich geförderten und der gemeinnützigen Forschung, aber unter bestimmten Bedingungen auch private Wirtschaftsunternehmen mit Bezug zu Forschung oder Wissenschaftsorganisationen.

Zur Finanzierung des WiN wurde zwischen dem DFN-Verein und der DBP ein Pauschalpreis von 9,7 Mio DM jährlich ausgehandelt (zum Vergleich: im Jahre 1988 bezahlten die bundesdeutschen Wissenschaftseinrichtungen 14 Mio DM für die Datenkommunikation an die DBP!). Für seine Mitglieder wurden vom DFN-Verein die folgenden festen Gebühren vereinbart:

* DM 60.000 pro Jahr incl. MwSt. für einen 64 Kb/s-Anschluß
* DM 15.000 pro Jahr incl. MwSt. für einen 9,6 Kb/s-Anschluß.

Das WiN ist für eine Gesamtdurchsatzrate von 50 GByte pro Monat bei zunächst maximal 125 Anschlüssen mit je 64 Kb/s und 110 Anschlüssen mit je 9,6 Kb/s ausgelegt. Steigender Nachfrage wird die DBP-T durch Ausbau des WiN Rechnung tragen.

Der WiN-Anschluß des ZIB weist eine Übertragungsgeschwindigkeit von 64 Kb/s auf bei 100 logischen Kanälen, d.h. es können maximal 100 Verbindungen gleichzeitig auf einer Anschlußleitung bestehen.
Er hat die Rufnummer 45 050 331*sss* (3-stellige Subadresse *sss*, Vermittlungsknoten Berlin). Folgende Anschlüsse sind betriebsbereit:

45 050 331 000 Cyber 825 - NOS/BE Dialog
45 050 331 001 Cyber 825 - NOS/BE RJE/FT
45 050 331 002 Cyber 930 - NOS/VE RJE/FT/Dialog
45 050 331 031 SUN 3/260 serv01 - Mail/Dialog
45 050 331 033 SUN 3/260 serv03 - eLib.

Mit der Inbetriebnahme des WiN-Anschlusses im ZIB im März 1990 wurde der Zugang zum DATEX-P-Netz der DBP vom bisherigen direkten Anschluß auf den WiN-Anschluß verlegt. Statt einer Übertragungsgeschwindigkeit von 9,6 Kb/s bei 30 logischen Kanälen stehen jetzt 64 Kb/s und 100 logische Kanäle zur Verfügung. Bei der Verwendung von Rufnummern ist der Anwender selbst dafür verantwortlich, nach Bekanntwerden der neuen WiN-Rufnummer diese auch zu verwenden. Dies gilt insbesondere für Verbindungen zu DATEX-P-Teilnehmern, die kostenpflichtig waren, seit Verfügbarkeit des WiN-Anschlusses der Gegenstelle aber kostenlos durchgeführt werden können.

1.4.2 DFN-Dienste

Im Rahmen des DFN werden folgende Dienste angeboten:

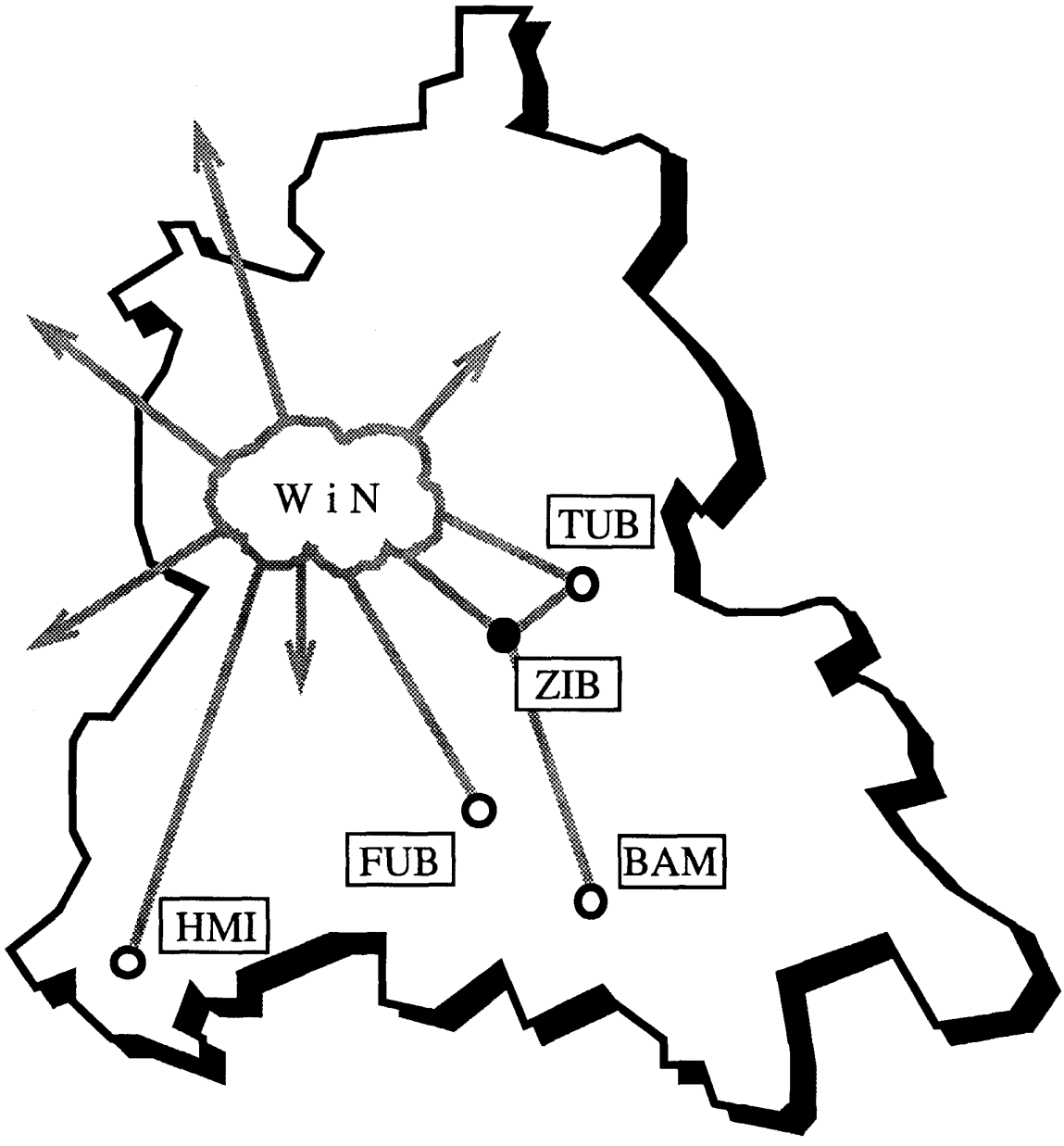
Remote Job Entry (RJE)

Der RJE-Dienst überträgt Ein- und Ausgabedateien zwischen den am Verbundsystem angeschlossenen Rechenanlagen. Die von einem Job erzeugte Ausgabedatei wird zu einer der angeschlossenen Rechenanlagen übertragen; im Regelfall ist dies die Anlage, von der aus der Job abgeschickt wurde.

Die Rechner des ZIB werden über folgende Adressen angesprochen:

Rechner	DFN-Adresse	WiN-Adresse
CRAY X/MP	B ZIB12	45050331002
CYBER 825	B ZIB01	45050331001

Berliner Wissenschaftsnetz (X.25)



Dienste: RJE, FT, X.400, X.29

Legende: 64Kb/s

CYBER 930 B ZIB02 45050331002

(die zwei Leerzeichen hinter B sind signifikant!)

Einzelheiten über das Absenden eines Batchjobs von einer dieser Anlagen erfahren Sie beim jeweiligen Rechenzentrum. Beispiele und Erklärungen zum Abschicken von Jobs für die CRAY von NOS/BE- oder VMS-Anlagen finden Sie in Kapitel 5.1.3. An den Vorrechnern des ZIB stehen folgende Informationen zur Verfügung:

DOC,DFN,RJEVE	Remote Job Entry an der CYBER 930 unter NOS/VE
DOC,DFN,RJEVEBE	Beispiele für Remote Job Entry zwischen der CYBER 930 unter NOS/VE und der CYBER 825 unter NOS/BE
DOC,DFN,RJECRY	Remote Job Entry zur CRAY X/MP
DOC,BERNET,RJE	Remote Job Entry an der CYBER 825 unter NOS/BE

File Transfer Dienst

Mit Hilfe des FT-Dienstes können codierte Dateien von einem entfernt stehenden Rechner geholt oder zu diesem gebracht werden.

Die Rechner des ZIB werden über folgende Adressen angesprochen:

Rechner	DFN-Adresse	WiN-Adresse
CYBER 825	B ZIB01	45050331001
CYBER 930	B ZIB02	45050331002

Die CRAY ist nicht direkt mit dem File-Transfer-Dienst erreichbar.

An den Vorrechnern des ZIB stehen folgende Informationen zur Verfügung:

DOC,DFN,FTVE	Filetransfer an der CYBER 930 unter NOS/VE
DOC,BERNET,FT	Filetransfer an der CYBER 825 unter NOS/BE
DOC,BERNET,FTINSI	Beispiele für den File-Transfer-Dienst zwischen CD unter NOS/BE und VAX unter PCS-MUNIX
DOC,BERNET,FTVMS	Beispiele für den File-Transfer-Dienst zwischen CD unter NOS/BE und VAX unter VMS

Passiver Dialog

Unter passivem Dialog versteht man die Möglichkeit, den betreffenden Rechner von außen her über die von der Deutschen Bundespost eingesetzten X.28/X.29-Protokolle anzusprechen. Der Benutzer kann von folgenden Geräten aus auf die ZIB-Rechner zugreifen:
- Terminals an einem Package Assembler Disassembler (Hardware-PAD), der direkt an eine Untervermittlung am WiN angeschlossen sind;
- Terminals an einem Rechner, der über den aktiven Dialog (Software-PAD) verfügt;
- Terminals, die direkten Zugang zum DATEX-P-Dienst der Deutschen Bundespost besitzen.

Sowohl die CYBER 930 als auch die CYBER 825 sind über den passiven Dialog von außen ansprechbar:

CYBER 930 45 050 331002
CYBER 825 45 050 331000

Aktiver Dialog

Unter aktivem Dialog (Software-PAD) versteht man die Möglichkeit, von einem am eigenen Rechner installierten Dialoggerät über diesen Rechner den X.25-Ausgang und über das DATEX-P-Netz bzw. des WiN der DBP-T den passiven Dialog eines entfernten Rechners zu verwenden. Fragen Sie in Ihrem Rechenzentrum nach, ob Ihr Rechner über den aktiven Dialog verfügt.

Rechner im DFN:

Innerhalb des Norddeutschen Vektorrechnerverbundes sind u.a. folgende Rechenanlagen im DFN erreichbar (Quelle: DFN-Informationssystem, Stand April 1990):

BERLIN:

Institution	Rechner	Dienste	DFN-Adresse
Bundesanstalt für Material-			
forschung und -prüfung	VAX 8600	R F D	B BAM03[2)]
Fritz-Haber-Institut	VAX 8600	R F D	B FHI01
Hahn-Meitner-Institut Berlin GmbH	S 7.890	R F D	B HMI02
Hahn-Meitner-Institut Berlin GmbH	S 7.890	R F	B HMI04
Hahn-Meitner-Institut Berlin GmbH	VAX 11/785	R F D	B HMI13
Hahn-Meitner-Institut Berlin GmbH	VAX 11/780	R F D	B HMI21
Freie Universität Berlin:			
- Fachbereich Geowissenschaften	MicroVAX	R F	B MET01
- Fachbereich Pharmazie	MicroVAX	R F	B PHA01
- Fachbereich Physik	VAX 11/780	R	B PHY01
- Zentraleinr. Datenverarbeitung	CDC 170-850	R F D	B FUB02
- Zentraleinr. Datenverarbeitung	MicroVAX	R F D	B FUB04
- Zentraleinr. Datenverarbeitung	S 7.550	R F D	B FUB05
Technische Universität Berlin:			
- Fachbereich 6 (Iwan N. Stranski-Inst.)	PCS CADMUS	R F D	B INS01
- Fachbereich 7 (Bauingenieurwesen)	PCS CADMUS	R F	B SBK01
- Zentraleinrichtung Rechenzentrum	CDC180-960 (BE)	R F D	B TUB01
- Zentraleinrichtung Rechenzentrum	CDC180-960 (VE)	R F D	B TUB02[1)]
- Zentraleinrichtung Rechenzentrum	CDC170-830 (VE)	R F D	B TUB03[1)]
- Zentraleinrichtung Rechenzentrum	S 7.551	R F D	B TUB05

NIEDERSACHSEN:

Institution	Rechner	Dienste	DFN-Adresse
Bundesanstalt für Geowiss. und Rohstoffe	VAX 8600	R	H BGR01
Gesellschaft für Strahlenforschung	VAX	R F	BS GSF05[3)]
Gesellschaft für wissenschaftliche DV	IBM 3090	R F	GO GWD02
Gesellschaft für wissenschaftliche DV	VAX 8650	R F	GO GWD52
Max-Planck-Institut für Aeronomie	VAX	R	GO MIO01
Max-Planck-Institut für Strömungsforschung	MicroVAX	R	GO MSF01
Technische Universität Braunschweig	Amdahl 970	R F	BS RTU01

Technische Universität Clausthal	VAX	R F	CLZRTU01
Regionales Rechenzentrum Hannover	MicroVAX	R F D	H RRZ10
Regionales Rechenzentrum Hannover	CDC 180-990	R F D	H RRZ21
Regionales Rechenzentrum Hannover	CDC 180-990	R F D	H RRZ22
Regionales Rechenzentrum Hannover	VP200	R F D	H RRZ41
Universität Hannover IFW	VAX	R F D	H RRZ11
Universität Hannover IFM	MicroVAX	R F D	H RRZ12

SCHLESWIG-HOLSTEIN:

Institution	**Rechner**	**Dienste**	**DFN-Adresse**
Rechenzentrum Universität Kiel	VAX 750	R F D	KI UNI04
Rechenzentrum Universität Kiel	S 7.760	R F D	KI UNI05
Rechenzentrum Universität Kiel	VAX 8550	R F D	KI UNI20
Rechenzentrum Universität Kiel	CRAY X-MP	R F	KI UNI21
Universität Kiel Institut für Meereskunde	VAX 8550	R F	KI UNI80
Universität Kiel Institut für Meereskunde	VAX 11/780	R F	KI UNI81

Erläuterung:

* Alle Rechner mit nicht gekennzeichneten Adressen sind über das Wissenschaftsnetz *WiN* erreichbar;
* R bedeutet RJE-Dienst,
* F bedeutet Filetransfer-Dienst,
* D bedeutet Dialog-Dienst.

[1] diese Rechner sind z.Zt. unter NOS/VE für Benutzer nicht zugänglich.

[2] bedeutet, daß der entsprechende Rechner mit den Rechnern des ZIB über Standleitung erreichbar ist.

[3] bedeutet, daß der entsprechende Rechner mit den Rechnern des ZIB **nicht** über das Wissenschaftsnetz *WiN* erreichbar ist; bei der Benutzung fallen also für den Benutzer DATEX-P-Übertragungsgebühren an.

Die DFN-Adresse dient zur eindeutigen Bezeichnung der Datenverarbeitungsanlage. Die ersten drei Zeichen stellen eine Gebietsbezeichnung dar, die von der Kennzeichnung für Kraftfahrzeuge übernommen worden ist; die (entweder ein oder zwei) Leerzeichen in dieser Adresse sind daher signifikant. Die folgenden drei Zeichen bilden eine Institutskennung, während die letzten zwei Ziffern die Unterscheidung zwischen verschiedenen Datenverarbeitungsanlagen des Instituts ermöglichen.

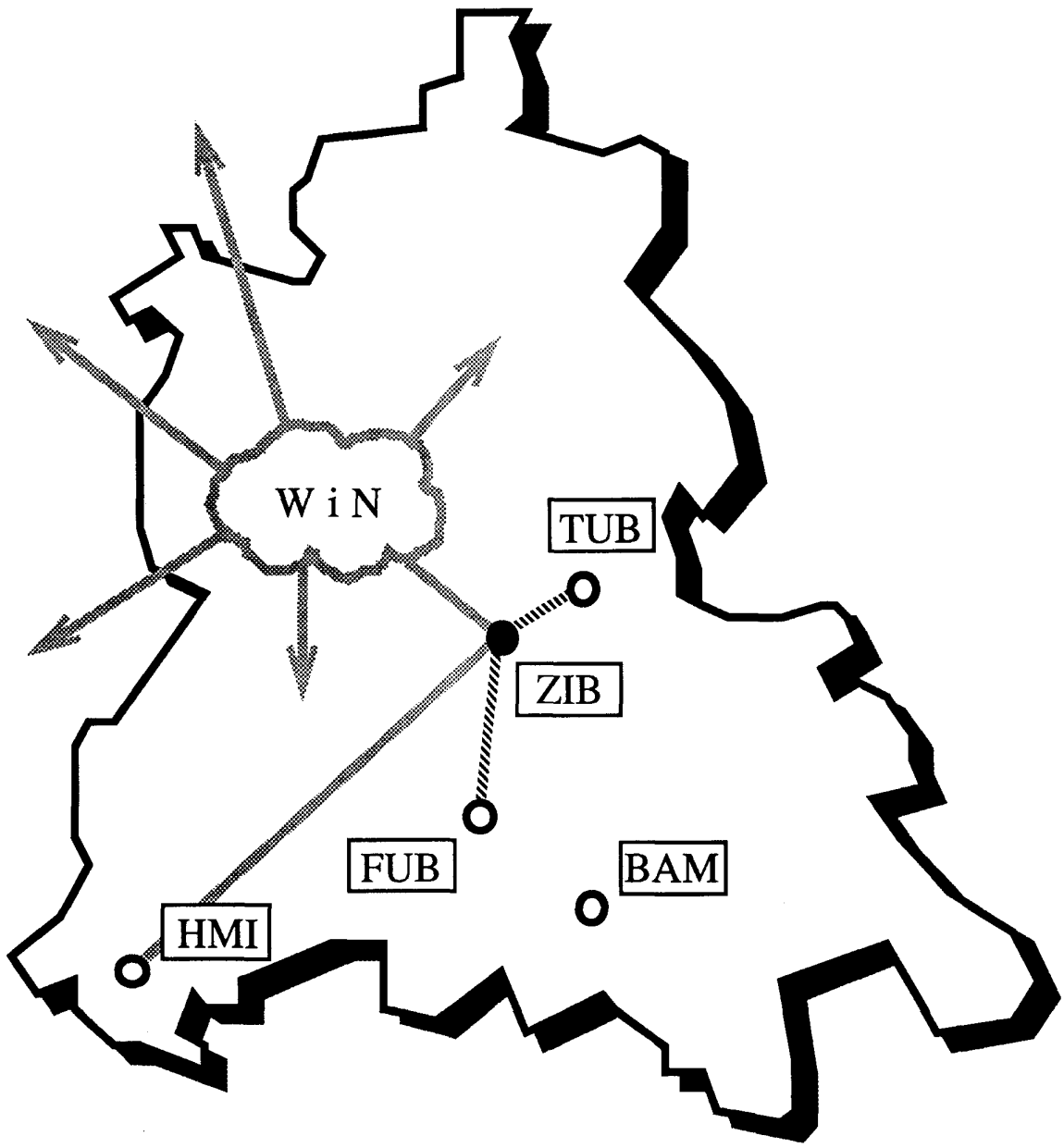
1.4.3 Ethernet-Anschlüsse

Die CRAY X-MP des ZIB sowie die UNIX-Workstations sind über *Ethernet* verbunden (Standard nach IEEE 802.3). Auf dem Ethernet fließen die Daten mit einer Transferrate von maximal 10 Mb/s. Das lokale Ethernet des ZIB ist mit den lokalen Netzen in der Freien Universität Berlin (2 Mb/s), der Technischen Universität Berlin (2 Mb/s) und dem Hahn-Meitner-Institut (64 Kb/s) direkt verbunden. Außerdem bestehen über das WiN Verbindungen zu weiteren an das WiN angekoppelten lokalen Netzen, z.B. die der Niedersächsischen Universitäten.

1.4.4 Internet-Dienste

Ursprünglich ist das Internet aus dem vom amerikanischen Verteidigungsministerium finanzierten ARPANET (Advanced Research Projects Agency Network) hervorgegangen. Obwohl man immer von dem "Internet" spricht, als ob es ein einziges Netzwerk wäre, besteht es aus verschiedenen Netzen, die unterschiedliche physikalische Transportmechanismen verwenden und auch verschiedene Administrationen besitzen. Allen am Internet beteiligten Netzwerken gemeinsam ist das Netzwerkprotokoll IP (Internet Protocol), die darauf

Berliner Wissenschaftsnetz (Ethernet)



Dienste: TCP/IP, CDCNET

Legende: 2 Mb/s
64 Kb/s

basierenden Transportprotokolle TCP (Transmission Control Protocol) und UDP (User Datagram Protocol) und die Applikationsprotokolle. Meistens wird die Internet Protokollfamilie mit TCP/IP abgekürzt. Wichtige Teilnetzwerke des Internet sind:

ARPANET	ursprünglicher Kern des Internet
MILNET	militärischer Teil des ARPANET
NSFnet	National Science Foundation Network
CSNET	Computer Science Network

Eine Einschätzung der effektiven Größe des Internet ist schwierig, da es keine zentrale Verwaltung der Rechnernamen gibt; es sind aber sicher mehrere 10.000 Computer via Internet erreichbar. Da es keinen namhaften Hersteller gibt, der TCP/IP nicht unterstützt, kann alles von einem PC bis hin zu einem Supercomputer am Internet angeschlossen sein.

Die technische Koordination des Internet übernimmt in den USA das Internet Activities Board (IAB) und die Verwaltung das Network Information Center (NIC) (z.B. werden IP Netzwerkadressen vom NIC verwaltet).

Die Normendokumente des Internet sind die RFC's (Request For Comment); diese können als Textdatei von verschiedenen Quellen bezogen werden. Regelmäßig wird ein RFC publiziert, der eine Liste der aktuellen offiziellen Protokolle beinhaltet.

Die drei Grundanwendungen, die auf dem Internet unterstützt werden, sind

FTP	Das *File Transfer Protocol* erlaubt es, Dateien zwischen Computern zu übertragen. Da es viele verschiedene Dateiformate gibt, können gewisse Umsetzungen (z.B. Zeilenende) automatisch gemacht werden.
SMTP	Das *Simple Mail Transfer Protocol*, mit dem zwischen zwei Computern elektronische Post ausgetauscht wird. Im Gegensatz zu X.400 oder UUCP (UNIX-UNIX-Copy) wird dabei immer eine direkte Verbindung zum empfangenden Computer aufgebaut. Die Adressierung und das Format der übertragenen Post wird im RFC822 festgelegt.
TELNET	erlaubt den interaktiven Zugang zu Computern am Netzwerk. Häufig werden sogenannte *Terminal Server* verwendet. Dies sind Geräte mit mehreren Terminalschnittstellen und einem Netz-Anschluß. Der Benutzer kann dann auch von einfachen Terminals mit TELNET einen beliebigen Host am Netzwerk erreichen.

Eine Beschreibung der zugehörigen Kommandos findet man in Kapitel 6. Neben diesen Grundanwendungen gibt es viele andere Applikationen, die auf TCP/IP basieren:

Anonymous FTP	hierbei handelt es sich um Programm- oder Dokumentenarchive, auf die mit FTP zugegriffen werden kann. Da FTP ein Konto auf der Server-Maschine bedingt, richtet man ein Konto ein, dessen Name und Passwort öffentlich bekanntgegeben werden. Zum Beispiel sind alle RFC's via Anonymous FTP erhältlich.
FINGER	Benutzerinformation
NNTP	*Network News Transfer Protocol*: Usenet News
TFTP	*Trivial File Transfer Protokol*: Dateitransfer
POP2	*Post Office Protocol*: Mailbox Server

Internet-Dienste im ZIB

Auf den Rechnern des ZIB werden folgende Internet-Dienste mit den zugehörigen Adressen angeboten:

Rechner	symb. Name	Internet-Adresse		Dienste	
CRAY X-MP:	*cray*	130.73.128.2 über *serv01*	FTP		SMTP[1)]
	cray	130.73.192.4 über *serv02*	FTP		SMTP[1)]
UNIX-front-end	*ufer*	130.73.108.21	FTP	TELNET	SMTP
CD CYBER 930:	*zib02*	130.73.100.2	FTP	TELNET	

[1)]: Auf der CRAY steht nur der aktive SMTP-Dienst zur Verfügung.

Mit dem lokalen Netz des ZIB sind zur Zeit (April 1990) folgende Netze verbunden:

Institution	Internet-Adresse	Art der Verbindung
ZIB	130.73.x.x	
FUB	130.133.x.x	2Mb/s HfD
TUB	130.149.x.x	2Mb/s HfD
HMI	134.30.x.x	64Kb/s HfD
UNIDO	129.217.x.x	64Kb/s WiN

1.4.5 CDCNET-Dienste

Die CRAY X-MP des ZIB ist über die CYBER 930 in das CDCNET des ZIB integriert. CDCNET benutzt NAM/VE, ein auf den Ethernet-Protokollen (Standard nach IEEE 802.3) basierendes Protokoll der Fa. Control Data. Aus der Sicht des CDCNET ist dieser Verbund *ein* Netz; die im folgenden beschriebenen Dienste beziehen sich jeweils auf dieses gesamte CDCNET.

An Diensten stehen Dialogzugang zu den Rechnern des CDCNET sowie Jobtransfer (Queuefile Transfer Facility - QTF) und Dateitransfer (Permanentfile Transfer Facility - PTF) zwischen den Rechnern zur Verfügung. Die Dienste QTF und PTF stehen nur zwischen NOS/VE-Rechnern zur Verfügung.

Die Rechner im CDCNET des NVV (Norddeutscher Vektorrechnerverbund) sind über folgende Namen erreichbar:

	Rechner	System	Bezeichnung
ZIB	CYBER 930	NOS/VE	ZIB02
ZIB	CRAY X-MP	UNICOS	ZIB_CRAY
ZIB	UNIX front end	UNIX	ZIB_UFER
FUB	CYBER 850	NOS/VE	FUB02
FUB	MicroVAX	VMS	FUB06
TUB	CYBER 960	NOS/VE	TUB02
RRZN	CYBER 990	NOS/VE	CDC1 (vorläufige Bezeichnung)
RRZN	CYBER 990	NOS/VE	CDC2 (vorläufige Bezeichnung)

Dialog-Dienst

Jedes am CDCNET angeschlossene Terminal erlaubt durch Aufruf des Kommandos

CREATE_CONNECTION name
(CREC)

einen Verbindungsaufbau zu einem Rechner am CDCNET. Auch Benutzer, die über den X.29-Dienst (PAD-Dienst, siehe Kapitel 1.4.2) eine Verbindung zum CDCNET erhalten haben, können sich zu den übrigen Rechnern im Netz weiterschalten.

Stationen und Drucker

Stationen und Drucker, die an irgendeiner Stelle in einem Teilnetz des CDCNET angeschlossen sind, können von allen am CDCNET angeschlossenen Rechnern angesprochen werden.

Namen einiger Drucker oder Stationen:

ZIB-Standarddrucker: ZIB_AUTOMATIC
FUB-Standarddrucker: FUB_FACIT
FUB-Standarddrucker: FUB_AUTOMATIC

Jobtransfer

Von jedem am CDCNET angeschlossenenn NOS/VE-Rechner kann mit dem üblichen SUBMIT-Kommando

```
SUBMIT_JOB dateiname
(SUBJ)
```

eine Datei als ein Job an einem anderen Rechner des CDCNET zur Ausführung gebracht werden; die Jobausgabe kann zu einem weiteren Rechner oder einem beliebigen Drucker am CDCNET transportiert werden.

Mit Hilfe des Kommandos

```
SUBMIT_CRAY_JOB dateiname
(SUBCJ)
```

können von jeder NOS/VE-Anlage des CDCNET Jobs direkt zur CRAY X-MP des ZIB geschickt werden; die Druckausgabe erfolgt wieder auf einem Rechner oder Drucker des CDCNET (siehe Kap. 5.1.2).

Dateitransfer

Von jedem am CDCNET angeschlossenem NOS/VE-Rechner kann zu einem anderen NOS/VE-Rechner eine Datei übertragen werden. Verwendet wird die übliche Kopierroutine

```
COPY_FILE  dat1  dat2
(COPF)
```

Die Dateinamen müssen jedoch vollständig "qualifiziert" sein; insbesondere legt der im CDCNET eindeutige Name der Family den externen Rechner fest. Mit Hilfe des Kommandos

```
CREATE_REMOTE_VALIDATION LOCATION=name  VALIDATION=string
(CRERV)
```

wird die Validierung für den Benutzer auf dem entfernten Rechner eingetragen, aber noch nicht geprüft.

Beispiel:

Der Benutzer setzt an der CYBER 930 des ZIB folgende Kommandos ab:

```
CRERV L=FUB V='LOGIN U=fuuserid PW=fupw'
COPF :FUB.fuuserid.datei datei
```

Die im Hauptkatalog des Benutzers auf dem NOS/VE-Rechner in der FU liegende Datei *datei* wird in den Hauptkatalog des Benutzers auf die CYBER 930 des ZIB kopiert.

1.5 Entgelte

Für die Benutzung der CRAY X-MP gelten die Entgeltordnungen der zulassenden Rechenzentren. Zur Zeit werden von Angehörigen der Universitäten in Berlin, Niedersachsen und Schleswig-Holstein noch keine Entgelte erhoben.

Bis zum Erlaß einer Entgeltordnung des ZIB gelten folgende Sonderregelungen für sonstige Benutzer:

Benutzer des Vektorrechners entrichten Entgelte für die verbrauchte Systemzeit, nämlich

Industriepartner des ZIB	6.000,- DM pro Stunde (zuzüglich MwSt)
Wissenschaftliche Einrichtungen außerhalb Berlins, Niedersachsens und Schleswig-Holsteins	1.000,- DM pro Stunde (zuzüglich MwSt)

In diesen Entgelten ist die Benutzung des jeweiligen Vorrechners inbegriffen. Eine Nutzung der Vorrechner zu anderen Zwecken als zum Zugang zum Vektorrechner ist nur im Rahmen von Forschungsprojekten des ZIB vorgesehen und muß im Einzelfall gesondert geregelt werden.

Die Entgeltordnung des ZIB regelt ausschließlich die Entgelte für die Nutzer des ZIB-Anteils am Vektorrechner. Für die Nutzer der Kontingente unserer Partner gelten weiterhin die Regelungen im Bereich des jeweiligen Partners.

Leitlinien der vorgeschlagenen Entgeltordnung für direkt vom ZIB zugelassene Benutzer sind:

* Für Projekte des Landes, der Bundeseinrichtungen und anderer von der öffentlichen Hand getragener Einrichtungen sowie Projekte, an denen ein besonderes Interesse des ZIB besteht, werden Entgelte erhoben, die an den Selbstkosten des ZIB orientiert sind (DM 1.500,- pro Stunde Systemzeit).

* Für alle sonstigen Projekte sind die Entgelte am Marktpreis orientiert (DM 6.000,- pro Stunde Systemzeit).

* Für alle Projekte wird ein Mindestentgelt von DM 90,- pro Quartal erhoben, das auf das nach obigen Regeln ermittelte Entgelt angerechnet wird.

* Im Rahmen von Sondervereinbarungen (insbesondere Kooperationsverträgen) mit anderen Einrichtungen kann der Präsident des ZIB Ausnahmeregelungen treffen, wobei im Grundsatz die Selbstkosten nicht unterschritten werden sollen.

Die Partner FUB und TUB des ZIB planen, ihre eigenen Entgeltordnungen so mit dem ZIB abzustimmen, daß für Benutzer aus dem Berliner Raum das zu entrichtende Entgelt unabhängig von der zulassenden Stelle ist. Für Forschungsprojekte und Lehrveranstaltungen der Universitäten und Hochschulen des Landes Berlin soll auch künftig kein Entgelt erhoben werden.

1.6 Betriebszeiten der CRAY X-MP

Die CRAY X-MP sowie beide Vorrechner werden rund um die Uhr betrieben; Operateure stehen

Montag bis Freitag von 6^{30} bis 18^{00} Uhr

zur Verfügung. Bandgeräte können nur bis 17^{30} Uhr benutzt werden. Für Wartungsarbeiten wird der Benutzerbetrieb zu folgenden Zeiten unterbrochen:

CRAY X-MP: montags 7^{00} - 10^{00} Uhr

CYBER 825: jeden zweiten Montag 7^{30} - 9^{30} Uhr

CYBER 930: nach Bedarf (ca. viermal pro Jahr) montags 7^{00} - 10^{00} Uhr.

Darüber hinausgehende absehbare Einschränkungen werden im Kopf jeder Jobliste (Header) sowie beim Beginn des Dialogs (Startup-Prozedur) den Benutzern mitgeteilt. Der aktuelle Betriebszustand kann über den Anrufbeantworter des ZIB, Rufnummer (030) 89604 167 abgefragt werden.

1.7 Adressen

Anschriften und Telefonnummern

Postanschriften:	Telefonnummern:
Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) Bereich Anlagenbetrieb Heilbronner Str. 10 1000 B E R L I N 31	Sekretariat: ..... (030) 8 96 04-131 Leitung: Hr. Gottschewski ..... 130 Abteilungen: CRAY/CD: Hr. Busch ..... 135 SUN : Hr. Kujawa ..... 150 Maschinensaalleiter: Hr. Götz 160
Freie Universität Berlin Zentraleinrichtung für Datenverarbeitung (ZEDAT) Fabeckstr. 32 1000 B E R L I N 33	Sekretariat: ..... (030) 838-6055 Leitung: Hr. Giedke ..... 4215 Beratung: Bereich CD: ..... 4206 SIEMENS: ..... 6029 Benutzerverwaltung: 6069
Technische Universität Berlin Zentraleinrichtung Rechenzentrum (ZRZ) Einsteinufer 17 1000 B E R L I N 12	Sekretariat: ..... (030) 314-22703 Leitung: Hr. Gürtler ..... 24230 Beratung Raum EN 029: ..... 25253 Rechnerbetrieb: Hr. Brodnicki ..... 24232 CD-Konsole 24236

Universität Kiel
- Rechenzentrum -
Olshausenstraße 40

2300 KIEL 1

Betreuung der CRAY-Benutzer:
Hr. Dr. Mordhorst ..... (0431)880-2766
Geschäftszimmer ..... -2768

Hahn-Meitner-Institut
Berlin GmbH
RZ D1
Glienicker Str. 100
Postfach 39 01 28

1000 BERLIN 39

Benutzerberatung:
Hr. Weidenhammer ..... (030) 8009-2596
Hr. Diemer ..... -2593

Regionales Rechenzentrum
für Niedersachsen (RRZN)
Universität Hannover
Schloßwender Str. 5

3000 HANNOVER 1

Telefon: ..... (0511) 762-2883
Betreuung der CRAY-Benutzer:
Hr. Buschmann ..... -4667
Hr. Fischer ..... -5132
Benutzerberatung: ..... -4737

Gesellschaft für wissen-
schaftliche Datenver-
arbeitung mbH Göttingen
Am Faßberg

3400 GÖTTINGEN

Telefon: ..... (0551) 201-
Betreuung der CRAY-Benutzer:
Dr. Weberpals ..... -552
Benutzerberatung: ..... -

Rechenzentrum
TU Braunschweig
Hans-Sommer-Str. 65
Postfach 3329

3300 BRAUNSCHWEIG

Telefon: ..... (0531) 391-
Betreuung der CRAY-Benutzer:
Dr. Busch ..... -5517
Benutzerberatung:
Dr. Schüle ..... -5542

Bundesanstalt f. Material-
forschung und -prüfung
Unter den Eichen 87

1000 BERLIN 45

Telefon: ..... (030) 8104-1
Betreuung der CRAY-Benutzer
und Benutzerberatung:
Dr. Müller ..... -7239

2. Dateien

Hinweis: Die folgenden Ausführungen gelten ohne Einschränkungen für Betriebssysteme, die UNIX System V von AT&T zur Grundlage haben. Die Kapitel 2.1 - 2.3 wurden mit geringfügigen Änderungen der Schrift "Einführung in UNIX" von H. Alt und M. Mitchelmore (LRZ München) entnommen.

2.1 Dateiarten

Eine Datei ist unter UNICOS eine Folge von Bytes; zumindest gilt dies für normale Dateien und Kataloge. UNICOS kennt vier Dateiarten:

* normale Dateien
* Dateikataloge
* Gerätedateien
* Pipes

Eine normale Datei entspricht konventionellen, aus anderen Betriebssystemen bekannten Dateien. In diesem Sinn kann eine gewöhnliche Datei beliebig Daten aufnehmen, beispielsweise Programme und Texte. UNICOS erwartet keinerlei spezielle Strukturierung einer Datei in Sätze, Blöcke, Sektoren usw. . Vom anwendungsbezogenen Standpunkt aus gesehen hat natürlich auch in UNICOS jede Datei eine interne Struktur; so besteht eine Textdatei beispielsweise aus einer Folge von Zeichen mit einer übergeordneten Zeilenstruktur, wobei die Zeilenenden durch besondere Zeilenende-Zeichen vermerkt sind.

Dateikataloge sind Dateien, die entweder leer sind, also nur einen Verweis auf sich selbst und auf ihren Vaterkatalog enthalten, oder aber Verweise enthalten auf weitere Dateien, die selbst Kataloge sein dürfen. Die Verweisstruktur ist hierbei hierarchisch. Die Anzahl der Einträge ist beliebig, aber natürlich durch die Größe des Datenträgers limitiert. Der Eintrag einer Datei besteht aus der systeminternen Knotennummer (i-node number) und dem Dateinamen. Die Einträge eines Kataloges kann man mit dem Kommando *ls* anzeigen:

Beispiel:

```
$ls -u
text.1          text.2          upro
$
```

Gibt man dem *ls*-Kommando noch die Option *-a* (d.h. all) mit, so werden auch die sogenannten verdeckten Dateien angezeigt - das sind diejenigen Dateien, deren Name mit einem Punkt beginnt:

Beispiel:

```
$ls -ax
.          ..          text.1          text.2          upro
$
```

Die beiden jetzt zusätzlich angezeigten Einträge sind in jedem Katalog vorhanden. Dabei ist der Punkt "." ein Verweiseintrag auf den aktuellen Katalog selbst und ".." der Verweis auf den Vaterkatalog .

Will man sich den zur Zeit aktuellen Katalog anzeigen lassen, so benutzt man dazu das Kommando *pwd* (print working directory).

Beispiele:

```
$pwd
/u4/btluser1
$
```

Mit dem Kommando *mkdir newdir* kann man einen neuen Katalog mit dem Namen *newdir* erzeugen:

```
$mkdir newdir
$ls -F
newdir/          text.1          text.2          upro
$
```

Mit der Option *-F* zeigt *ls* an, ob eine Datei ein Katalog (angehängtes "/") oder ein ausführbares File (angehängter "*") ist.

Um von einem Katalog in einen anderen wechseln zu können, benutzt man das Kommando *cd dirname* (cd: change_directory).

Beispiel:

```
$cd newdir
$pwd
/u4/btluser1/newdir
$ls -a
.     ..
$cd ..
$pwd
/u4/btluser1
$
```

Mit dem Befehl *cd ..* wechselt man also in den Vaterkatalog!

Gerätedateien, auch *special files* genannt, sind Dateien, welche für physikalische Geräte stehen. Durch ihre den normalen Dateien syntaktisch gleiche und semantisch, soweit sinnvoll, identische Behandlung ergibt sich für den Benutzer kein Unterschied zwischen der Ein-/Ausgabe auf Dateien oder auf physikalische Geräte, und damit auch weitestgehende Geräteunabhängigkeit. Die meisten *special files* sind im Katalog */dev* eingetragen (z.B. */dev/lpt01* ist der Eintrag für einen Drucker; */dev/null* ist der Eintrag für das Nullgerät: jede Ausgabe darauf wird weggeworfen, jede Eingabe von dort liefert EOF).

Pipes sind systeminterne Dateien zur Intertaskkommunikation (siehe Kapitel 4.1.2).

2.2 Dateinamen und -pfade

Ein Dateiname darf aus bis zu 14 Zeichen bestehen, wobei alle Zeichen erlaubt sind, also auch solche, die nicht druckbar sind. Aus praktischen Gründen jedoch sollte man sich auf Buchstaben (Groß- und Kleinbuchstaben werden unterschieden), Ziffern und die Zeichen ". , _" beschränken. Dateinamen dürfen im Prinzip frei gewählt werden, allerdings sollten Namenskonventionen für die Verwendung bestimmter Produkte beachtet werden z.B.:

a	für Objektbibliotheken
c	für C-Quelltextdateien
f	für FORTRAN-Quelltextdateien

o für Objektdateien

p für PASCAL-Quelltextdateien

Die UNICOS-Dateistruktur ist hierarchisch oder umgedreht baumartig. Der Ausgangspunkt eines solchen Baumes ist die Wurzel *root*, die unter UNICOS mit "/" angegeben wird. Die Wurzel selbst ist ein Katalog, in dem Verweise auf weitere Dateien angegeben sind, die wiederum selbst Kataloge sein können. Auf diese Weise entsteht die Baumstruktur. Der Zugriffspfad *path name* gibt an, wie eine Datei, ausgehend von der *root*, erreicht werden kann. Der Pfadname der Datei *text.1* in obigem Beispiel ist also */u4/btluser1/newdir/text.1* . Man nennt einen solchen Pfad auch absoluten Pfad. Man kann überall dort, wo man Dateinamen verwendet, auch den gesamten Pfadnamen angeben. Auf diese Weise kann ein Name mehrmals in verschiedenen Katalogen verwendet werden, da die zugehörige Datei über den Pfad eindeutig bestimmt ist. Man kann auch mit einem relativen Pfadnamen arbeiten, der den Zugriffspfad vom aktuellen Katalog aus zur Zieldatei angibt.

Ein und dieselbe physikalische Datei kann in einem Dateibaum auch mehrere Namen besitzen, die auf verschiedenen Ästen liegen; diese müssen allerdings im selben *file system* (vgl. Kap. 2.5) liegen. Die Datei ist dann unter all ihren Namen ansprechbar. Diese Verweise verschiedener Dateinamen auf eine Datei bezeichnet man als *links;* sie werden mit dem Kommando *ln alter_Name neuer_Name* erzeugt.
Vorsicht: wenn eine Datei mit dem Namen *neuer_Name* bereits existiert, geht ihr Inhalt verloren! Ebenso wird bei den Kommandos *mv* und *cp* (s.u.) verfahren; man erhält keinen Hinweis vom System!

Beispiel:

```
$pwd
/u4/btluser1
$ln text.1 newdir/text.ln
$cd newdir
$ls -F
text.ln
$
```

Man kann einer Datei natürlich nicht nur mehrere Namen geben, sondern sie auch umbenennen, kopieren oder löschen. Zum Umbenennen einer Datei benutzt man das Kommando *mv alter_Name neuer_Name* (move), zum Kopieren das Kommando *cp alter_Name neuer_Name* (copy), zum Löschen von Dateien das Kommando *rm Datei_Name* (remove) und zum Löschen von Katalogen das Kommando *rmdir Dir_Name* (remove_directory).

Beispiel:

```
$pwd
/u4/btluser1/newdir
$rm test.ln
$ls -a
.     ..
$cd
$
```

Mit dem *rm*-Kommando werden also nicht nur Dateien, sondern auch Verweise gelöscht. Eine Datei wird übrigens erst dann gelöscht, wenn alle *links* gelöscht worden sind. Auf Systemebene bewirkt der *rm*-Befehl nur den Austrag des Katalogeintrages.

```
$cp text.2 newdir/text.2c
```

Beim *cp*-Befehl wird die kopierte Datei physikalisch neu angelegt, im Gegensatz zum *ln*-Befehl, der nur einen Verweiseintrag im entsprechenden Katalog vornimmt.

```
$mv text.2 brief
$ls -x
brief          newdir          text.1          upro
$rmdir newdir
rmdir: newdir: Directory not empty
$
```

Man kann einen Katalog nur dann löschen, wenn er leer ist, oder falls man die Option *-R* (Rekursiv) mit angibt.

```
$rmdir -r newdir
$ls -x
brief    text.1    upro
$
```

!!! Vorsicht !!! Bei der Angabe der Option *-R* beim *rmdir*-Kommando werden alle Dateien und auch alle weiteren Kataloge gelöscht, die im Dateibaum **unterhalb** des angegebenen Kataloges liegen! Ähnlich vorsichtig sollte man das Kommando *rm* handhaben: bei *rm -r abc ** (statt *abc**, also mit Leerzeichen vor dem Stern) werden absolut alle Dateien im aktuellen und allen darunter liegenden Katalogen gelöscht!!

Allgemein empfielt es sich (solange man noch nicht genügend Sicherheit im Umgang mit einem Kommando besitzt), vor dessen Benutzung die zugehörige Beschreibung zu studieren!

2.3 Dateiattribute und Zugriffsrechte

Dateien haben eine Reihe von Attributen, von denen der Benutzer die meisten allerdings nicht ständig sieht. Zu diesen Attributen gehören:

* Dateiname
* Datei-Zugriffspfad
* Zugriffsrecht (protection-bits)
* Dateilänge (in Bytes)
* Knotennummer (i-node number), eine eindeutige Nummer innerhalb eines Dateisystems
* Anzahl der Links auf die Datei
* Datum der letzten Änderung
* Dateityp (normale Datei, Katalog oder *special file*)
* Benutzernummer des Besitzers und seiner Gruppe

Für jede Datei gibt es drei Klassen von Benutzern, die sich in ihren Zugriffsrechten unterscheiden:

* Dateibesitzer (**u** = login user)
* Benutzer der gleichen Gruppe (**g** = group)
* Alle anderen Benutzer (**o** = other users)

Jeder Benutzer ist eindeutig über seine UID (User-Identification-Number) identifiziert. Diese Nummer ist in der Paßwort-Datei */etc/passwd* eingetragen. Außer dieser UID ist jedoch bei jedem Benutzereintrag noch eine GID (Group-Identification-Number) eingetragen. Mit diesem Eintrag wird die Zugehörigkeit des Benutzers zu einer Gruppe festgelegt. Die Gruppen selbst werden in der Datei */etc/group* definiert, in der zu jeder GID ein Gruppenname und die Namen der Gruppenmitglieder eingetragen sind. Eine Gruppe kann beliebig viele Mitglieder haben, ebenso kann ein Benutzer Mitglied beliebig vieler Gruppen sein.

Auf jede Datei gibt es drei Arten von Zugriffsrechten

* Lesen (r = read)

* Schreiben (w = write)

* Ausführen (x = execute)

und zwar jeweils für den Eigentümer, die Gruppe und für alle anderen. Insgesamt können also 9 Zugriffsrechte (protection bits) pro Datei gesetzt werden. Diese Zugriffsrechte gibt es für alle Dateien, also auch für *special files* und Kataloge. Mit der Vergabe von Rechten sollte aus Gründen des Datenschutzes und der Datensicherheit sehr vorsichtig umgegangen werden. Eine Spezialität stellen die Zugriffsrechte auf Kataloge dar. Um einen Katalog ansehen und durchsuchen zu dürfen, benötigt man die Ausführungsrechte auf den Katalog. Eine Datei löschen kann man jedoch genau dann, wenn man auf den Katalog Schreibberechtigung besitzt, unabhängig davon, ob man überhaupt Zugriff auf die zu löschende Datei besitzt oder nicht.

Umfassend informieren über die Attribute kann man sich wiederum mit dem *ls*-Kommando, und zwar mit den Optionen *-l* (long), und *-i* (i-node number).

Beispiel:

```
$ls -lai
total 5
203  drwxr-xr-x  2  btluser1  btl    512  Jul  26  14:55  .
172  drwxr-xr-x  9  root      wheel  512  Jul  24  13:42  ..
642  -rw-------  1  btluser1  btl    386  Aug   4   9:12  brief
732  -rw-r-----  1  btluser1  btl     78  Aug   9  17:01  text.1
993  -rwxr--r--  1  btluser1  btl    113  Sep   4  11:24  upro
$
```

Werden bei einem Kommando mehrere Optionen benutzt, so sind sie in der Regel direkt hintereinander zu schreiben.

Die Ausgabe des Kommandos *ls -lai* (auf anderen UNIX Systemen *ls -lagi*) bedeutet im einzelnen folgendes:

* In der ersten Zeile wird angegeben, wieviel Blöcke (ein Block entspricht 512 Worten zu 8 Bytes) die Einträge in diesem Katalog belegen (in diesem Fall 5). Darauf folgt eine Liste aller Dateien, wobei pro Datei eine Zeile Informationen ausgegeben wird.
* In der ersten Spalte steht die *i-node number* der Datei. Sie wird systemintern zur eindeutigen Identifikation der Datei benutzt.
* Im zweiten Block stehen die Dateiart und die 9 *protection bits*. Diese bedeuten im einzelnen folgendes:
  - In der ersten Spalte wird die Dateiart angezeigt, nämlich:
    * **d** Datei ist ein Katalog
    * **-** normale Datei
    * **b** special file blockorientiert
    * **c** special file characterorientiert
    * **p** FIFO-Puffer (named pipe special file)

- Die nachfolgenden 9 Bits zeigen die Zugriffsberechtigung an, und zwar
  - * r für die Leseberechtigung,
  - * w für die Schreibberechtigung,
  - * x für das Ausführungsrecht

  auf einer Datei. Dabei gelten
  - * die ersten drei Bits für den Eigentümer der Datei,
  - * die nächsten drei Bits für die Gruppe,
  - * die letzten drei Bits für alle anderen Benutzer.
* In der dritten Spalte steht die Anzahl der Verweise auf die Datei.
* In der vierten und fünften Spalte steht der Name des Eigentümers oder der zugehörigen Gruppe. Ist der Loginname des Eigentümers bzw. der Gruppenname nicht mehr bekannt (wenn z.B. ein Benutzer keine Berechtigung mehr hat und aus */etc/passwd* gelöscht wurde), so steht an der entsprechenden Stelle die UID bzw. die GID.
* In der sechsten Spalte wird die Dateilänge in Bytes angegeben.
* In den nächsten drei Spalten wird Datum und Uhrzeit der letzten Dateiänderung angezeigt.
* In der letzten Spalte steht der Name der Datei. Der Name des Kataloges selbst und dessen Vaterkatalog werden dabei mit "." und ".." abgekürzt.

Bis auf die *i-node number* kann der Benutzer alle Dateiattribute ändern, auch die Eigentumsrechte (mit dem Befehl *chown*) und die Gruppenzugehörigkeit einer Datei (mit *chgrp*); aber Vorsicht: *chown* kann man nicht selber rückgängig machen!

Die Namensänderung geschieht mit dem Kommando *mv*, wie bereits oben besprochen. Das Zugriffsdatum und die Dateilänge ändern sich natürlich bei jedem Editiervorgang, und die Anzahl der *links* ist ebenfalls jederzeit änderbar (*ln*-Kommando).

Mit dem Kommando *chmod* kann der Eigentümer einer Datei die Zugriffsrechte einer Datei ändern. Für dieses Kommando gibt es zwei verschiedene syntaktische Formen:

* chmod *ooo filename*

* chmod *permissionlist filename*

Im ersten Fall steht *ooo* für eine dreistellige Oktalzahl, die - als Binärzahl gelesen - genau dort die Einsen hat, wo das entsprechende Zugriffsrecht gesetzt sein soll.

Beispiel:

```
$chmod 640 brief
$ls -lai
total 5
203  drwxr-xr-x  2  btluser1  btl    512  Jul  26  14:55  .
172  drwxr-xr-x  9  root      wheel  512  Jul  24  13:42  ..
642  -rw-r-----  1  btluser1  btl    386  Aug   4   9:12  brief
732  -rw-r-----  1  btluser1  btl     78  Aug   9  17:01  text.1
993  -rwxr--r--  1  btluser1  btl    113  Sep   4  11:24  upro
$
```

Die Oktalzahl 640 lautet in Binärdarstellung 110 100 000, was genau den Zugriffsrechten (daher auch der Name protection bits) rw-r----- der Datei *brief* entspricht.

Manche Benutzer möchten vielleicht nicht unbedingt einen Satz *flags* in eine Oktalzahl umrechnen und werden deshalb die zweite Methode bevorzugen. Die *permission list* hat dabei folgende Form: Benutzerklasse +/- Rechte. Als Benutzerklasse ist außer den drei bekannten (*u, g* und *o*) noch die Klasse *a* (= all) möglich. Als Recht ist das bekannte *r, w* und *x*, auch kombiniert, möglich.

Beispiel:

```
$chmod g-r text.1
$chmod a+wx upro
$ls -lai
total 5
203  drwxr-xr-x  2  btluser1  btl    512  Jul  26  14:55  .
172  drwxr-xr-x  9  root      wheel  512  Jul  24  13:42  ..
642  -rw-r-----  1  btluser1  btl    386  Aug   4   9:12  brief
732  -rw-------  1  btluser1  btl     78  Aug   9  17:01  text.1
993  -rwxrwxrwx  1  btluser1  btl    113  Sep   4  11:24  upro
$
```

In diesem Beispiel wurden der Gruppe *btl* also das Leserecht auf *text.1* entzogen und allen Benutzerklassen das Schreib- und Ausführungsrecht auf die Datei *upro* gegeben.

Ausdrücklich sei an dieser Stelle noch einmal darauf hingewiesen, daß man mit der Vergabe von Rechten sparsam umgehen sollte. In obigem Beispiel kann jeder Benutzer nun die Datei *upro* ausführen, aber auch nach Belieben verändern, was nicht unbedingt im Sinne des Eigentümers sein könnte.

Bei neuer Anlage einer Datei wird diese automatisch vom System mit *protection bits* versehen. Diese Voreinstellung kann vom Benutzer auf seine eigenen Bedürfnisse hin geändert werden, und zwar mit dem Kommando *umask ooo. ooo* ist hierbei wieder eine Oktalzahl, die allerdings genau komplementär zum Kommando *chmod* wirkt: Die Bits, die in dieser Maske gesetzt werden, werden bei der Neuanlage einer Datei nicht gesetzt. Der *umask*-Befehl gilt bis zum Jobende bzw. bis zum nächsten *umask*.
Beispiel:

```
$umask 177
$
```

Jede von diesem Zeitpunkt an erzeugte Datei wird nur *read* und *write* Berechtigung für den Eigentümer besitzen, andere Benutzer haben überhaupt keine Zugriffsrechte. Für *umask* gilt im ZIB die Voreinstellung *027*. Aus Datenschutzgründen ist dies eine empfehlenswerte Voreinstellung.

2.4 Kommandos zur Bearbeitung von Dateiinhalten

Das Kommando *cat*

Das Kommando *cat* dient zur Ausgabe, zur Erzeugung bzw. zur Verknüpfung von Dateien. Im folgenden werden einige typische Anwendungsbeispiele gegeben:

a) Ausgeben einer Datei nach *stdout*:

```
cat datei1
```

b) Verknüpfen zweier Dateien und Ausgabe in eine dritte Datei:

```
cat datei1 datei2 > datei3
```

Dabei wird *datei3* bei Bedarf neu eingerichtet bzw. ohne Warnung überschrieben, wenn sie schon vorhanden war.

c) Die Datei *datei1* wird ans Ende von *datei2* kopiert:

```
cat datei1 > > datei2
```

d) Die Datei *datei1* wird, falls nötig, eingerichtet und mit den Zeilen *zeile 1 ... zeile n* gefüllt bzw. überschrieben. Das Kommando *cat* zusammen mit den Eingabedaten bezeichnet man auch als *here document*.

```
cat > datei1 < < %
        zeile 1
        .
        .
        zeile n
%
```

Das Kommando *head*

Das Kommando *head* dient dazu, den Anfang einer Datei nach *stdout* auszugeben.

Beispiel:

```
head -30 quelle.f
```

Es werden die ersten 30 Zeilen der Datei *quelle.f* nach *stdout* ausgegeben. Die Option -n für die Anzahl der Zeilen ist mit 10 voreingestellt.

Das Kommando *tail*

Mit dem Kommando *tail* für die Ausgabe eines Dateiendes hat man mehr Möglichkeiten; es hat die Form

```
tail sign[number][unit] file
```

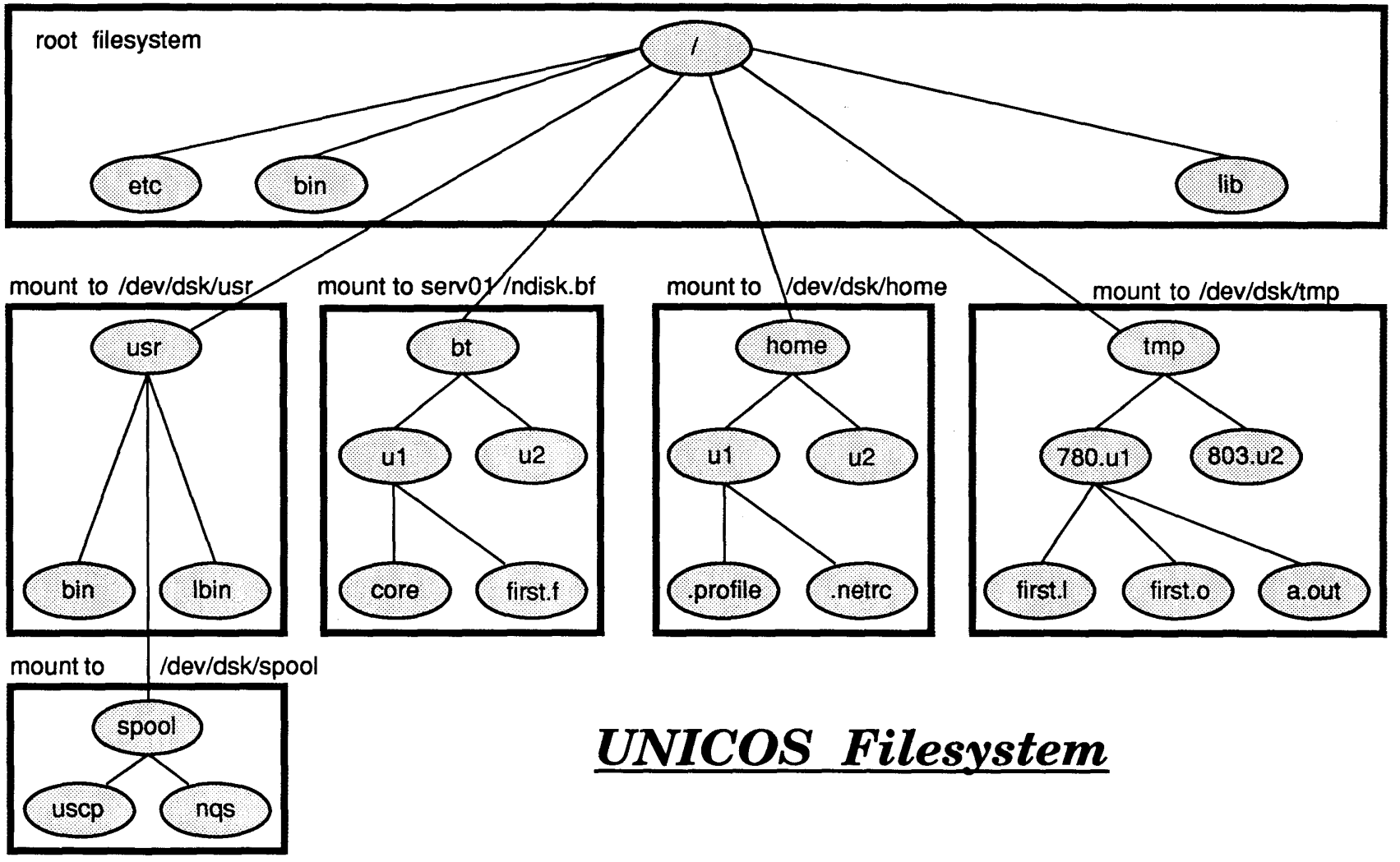
Die einzelnen Angaben haben folgende Bedeutung:

sign	'+' steht für die Zählung von Einheiten (*unit*) vom Anfang der Datei; '-' für die Zählung vom Ende der Datei, um die Startposition für die Ausgabe zu ermitteln.
number	Gibt die Anzahl von Einheiten für die Ermittlung der o.g. Startposition an (Default: 10).
unit	Einheit: l = Zeilen, b = Blöcke, c = Zeichen (Default: l).
file	Name der Datei.

Beispiel:

```
tail -15 quelle.f
```

Von Datei *quelle.f* werden die letzten 15 Zeilen nach *stdout* ausgegeben.



UNICOS Filesystem

2.5 UNIX Filesystem

In Beschreibungen von UNIX und UNIX-ähnlichen Betriebssystemen (z.B. UNICOS) wird der Begriff Filesystem mit zwei verschiedenen Bedeutungen verwendet:

Zum einen bezeichnet er die Gesamtheit aller im System verfügbaren Dateien und ihre Verwaltung (*root*-Katalog, Unterkataloge,...) auf physikalischen Datenträgern, zum anderen Gruppierungen von Dateien (z.B. nach Gesichtspunkten des Accounting oder der Optimierung von Zugriffen), die unabhängig von der physikalischen Organisation logische Einheiten bilden (*logische Datenträger*). Einige Aspekte dieser zweiten Bedeutung des Begriffs Filesystem innerhalb von UNICOS sind nachfolgend kurz aufgezählt:

* Ein Filesystem ist ein festgelegter Bereich auf Plattenspeichern, im Buffer Memory oder im SSD (solid-state storage device). Die Verwaltung geschieht in Einheiten von Blöcken (= Sektoren = 512 Worte = 4096 Bytes). Die konkrete Aufteilung des gesamten verfügbaren Hintergrundspeichers in bestimmte Filesysteme an der CRAY X-MP des ZIB ist dem Kap. 2.7 zu entnehmen.

* Ein Filesystem kann kleiner als ein physikalischer Datenträger sein, aber unter UNICOS auch Teile verschiedener physikalischer Datenträger logisch zusammenfassen.

* Einzelne Dateien müssen vollständig in ein Filesystem passen.

* Der *root*-Katalog bildet mit den Unterkatalogen der obersten Hierarchiestufe üblicherweise ein Filesystem.

* Der Zugriff auf ein Filesystem kann durch Administratorkommandos ermöglicht bzw. verhindert werden (*mount, umount*).

* Der wichtigste Punkt aus Benutzersicht: Das Kommando *ln* zur Definition von Aliasnamen ist nur innerhalb eines Filesystems zulässig. Bei der Überschreitung von Filesystemgrenzen muß statt dessen *assign* zusammen mit *env* verwendet werden (siehe Kap. 2.4.2).

2.6 Dateien in FORTRAN

2.6.1 Allgemeines

Im folgenden sind einige grundlegende Hinweise zusammengestellt, die bei Ein-/Ausgabevorgängen vom FORTRAN-Anwender unter UNICOS zu beachten sind. Die Ausführungen gelten, wenn nichts anderes gesagt ist, für beide unter UNICOS verfügbaren FORTRAN-Compiler CFT77 und CFT.

* Eine von einem FORTRAN-Programm für Ein-/Ausgabe verwendete Datei hat den Standardnamen *fort.u*, wobei *u* die Kanalnummer (unit; 0 < = u < = 101) bezeichnet. Die Kanalnummern 5 und 100 bzw. 6 und 101 sind bei Programmstart mit *stdin* bzw. *stdout* verbunden, ferner Kanalnummer 0 mit *stderr* (kein ANSI FORTRAN Standard).

* Bei Programmstart wird eine Datei von vorne bearbeitet; Dateipositionen bleiben über Programmschritte hinweg nicht erhalten; man kann jedoch in der OPEN-Anweisung des CFT77 über den Parameter POSITION das REWIND unterdrücken oder auch an das Ende der Datei positionieren.

* Werden in einem FORTRAN-Programm Dateien ohne explizites OPEN bearbeitet, dann gelten folgende Regeln:

  a) Beim ersten Lesezugriff auf Kanalnummer u muß die Datei *fort.u* im Arbeitskatalog vorhanden sein.

  b) Beim ersten Schreibzugriff auf Kanalnummer u wird die Datei *fort.u* im Arbeitskatalog angelegt oder, falls sie schon vorhanden war, ohne Warnung überschrieben.

* Ein FORTRAN-Programm, das mit einer Eingabedatei über Kanalnummer 5 und mit einer Ausgabedatei über Kanalnummer 6 arbeitet, kann ohne vorbereitende Kommandos folgendermaßen gestartet werden (beide Dateien im Arbeitskatalog):

  a.out < eingabe > ausgabe

* Dateinamen im FILE-Parameter des OPEN müssen so geschrieben werden, wie sie auf Kommandoebene festgelegt wurden (Groß-/Kleinschreibung!).

* In Dateinamen der Form *fort.u* müssen Kanalnummern u < 10 einstellig geschrieben werden, um vom FORTRAN-Laufzeitsystem erkannt zu werden.

2.6.2 Modifikation von Dateikenndaten

Für Ein-/Ausgaben über Kanalnummern ungleich fünf und sechs gibt es im wesentlichen zwei Verfahren, konkreten Dateien FORTRAN-Dateinamen bzw. sonstige Kenndaten zuzuordnen:

Wenn für eine zu bearbeitende Datei nur der vom FORTRAN-System geforderte Name *fort.u* vergeben werden muß, kann man die Datei entweder direkt mit *fort.u* benennen oder mit dem Kommando *ln* den gewünschten Dateinamen als *alias* festlegen, z.B.:

```
ln eingabe fort.9
```

Bei der Angabe von absoluten Pfadnamen ist zu beachten, daß Dateiname und *alias* im selben Filesystem liegen müssen. Liegen jedoch Dateiname und *alias* in verschiedenen Filesystemen (z.B. Dateiname im *home-directory* und *alias* in dem durch die Variable $TMPDIR festgelegten Arbeitskatalog) oder sollen für die Datei weitere Kenndaten wie z.B. Puffergröße oder Zugriffsmethode definiert werden, so sind die Kommandos *assign* und *env* notwendig.

Das Kommando *assign* hat folgende Form:

assign -a alias -b bs -c -n sz -p part -d bdr -s fs -t -V file

Parameter (Auswahl; default unterstrichen):

-a alias — *Aliasfile:* Name der Datei, für die weitere Kenndaten definiert werden sollen; muß bereits existieren; auch der volle Pfadname ist zulässig.

-b bs — *Buffer size:* Puffergröße in Blöcken zu 512 Worten.

-n sz — *Size:* Dateigröße in Blöcken zu 512 Worten; eine existierende Datei wird um *sz* Blöcke erweitert; die Reservierung des Speicherplatzes erfolgt erst beim OPEN.
4

-s fs — *File structure:* Dateistruktur bzw. Zugriffsmethode (Auswahl):

-s cos:	COS blocked - default für unformatierte E/A
-s u:	Undefined - Keine Pufferung, nur ganze Blöcke
-s sbin:	Standard binary - Standard-Binärdateien, mit Pufferung

file — *File:* Dateiname, in der Regel von der Form *fort.u* (vgl. 2.6.1)

Die mit dem Kommando *assign* definierten Dateikenndaten werden in zwei Schritten dem System mit dem Kommando *env* übermittelt:

* Im ersten Schritt wird über die Systemvariable FILENV ein *environment file*, das die Kenndaten aufnimmt, eingerichtet oder fortgeschrieben; mehrere *env*-Kommandos auf dieselbe Datei sind möglich.

* Im zweiten Schritt wird beim Programmstart das *environment file* angesprochen.

Beispiel:

```
cd $TMPDIR
env FILENV=asgn_1 assign -a $HOME/input fort.9
env FILENV=asgn_1 assign -a output fort.10
cft77 $HOME/prg.f
segldr prg.o
env FILENV=asgn_1 a.out > $HOME/results
cd
```

2.7 Dateiverwaltung auf der CRAY X-MP im ZIB

Auf der CRAY X-MP des ZIB sind eine Reihe von Filesystemen eingerichtet; ihre Namen, Größen und Verteilungen auf die im ZIB installierten Magnetplatten sowie den SSD (solid-state storage device, siehe 2.5) sind der Abbildung auf Seite 2-13 zu entnehmen.

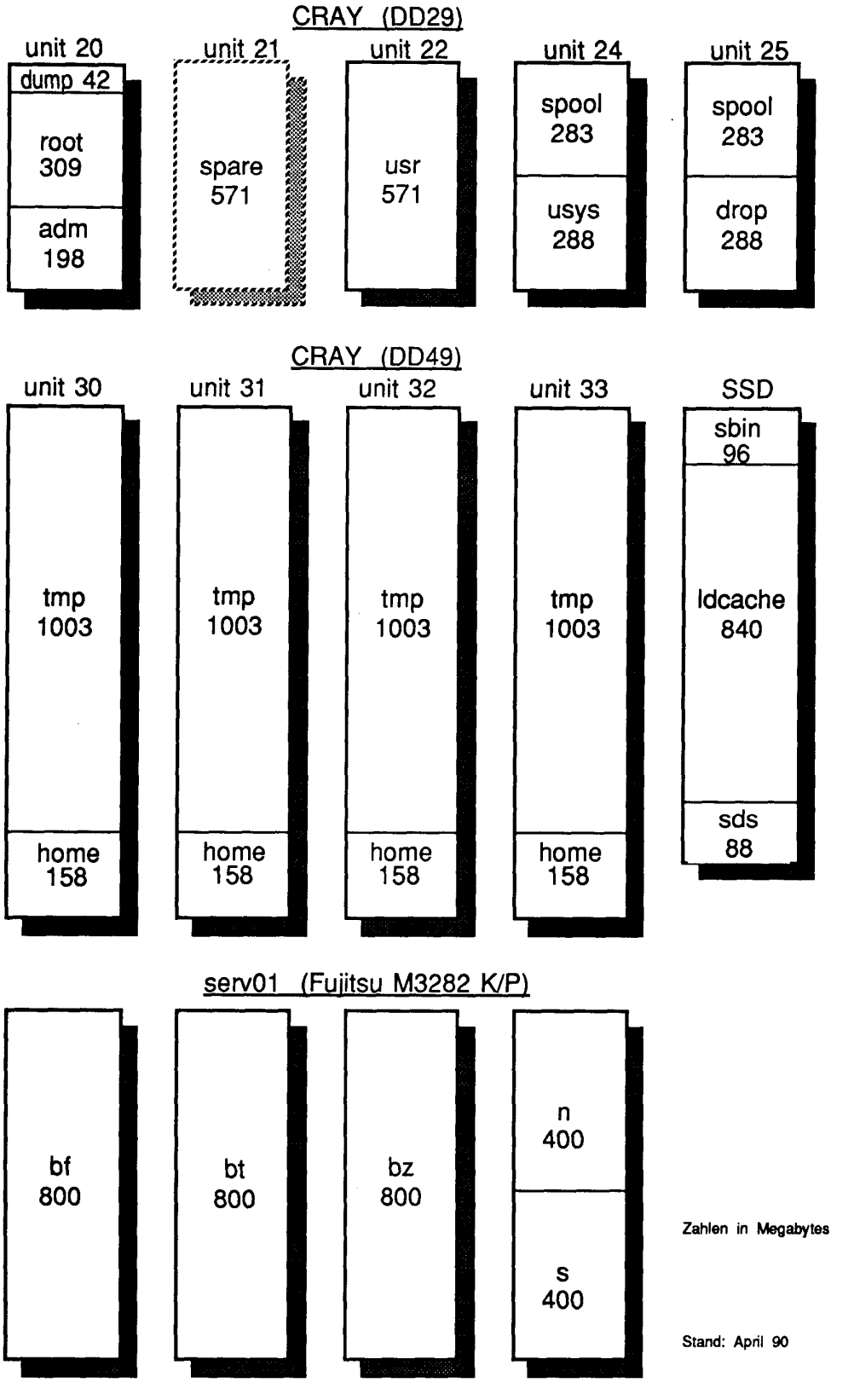
Die folgenden Filesysteme enthalten Systemdateien, die natürlich auch für den Benutzer vorhanden sind, auf die der Benutzer jedoch keinen allgemeinen Zugriff hat:

root	/: Wurzel aller Kataloge; enthält die Kataloge, die keinen anderen Filesystemen zugeordnet sind.
usr	/usr: weitere Systemkataloge, enthält Kataloge, die nicht in den folgenden Filesystemen abgelegt sind.
adm	/usr/adm: Statistikdaten der CRAY.
spool	/usr/spool: Ein- und Ausgabewarteschlangen.
drop	/drop: Systemkatalog zum Auslagern von in Ausführung befindlichen Programmen, enthält einen *cache* im SSD.
sbin	/sbin: Kopie von /bin mit den wichtigsten UNICOS-Dienstprogrammen, liegt zum schnelleren Laden im SSD.
usys	/usys: Benutzerkatalog für Systemverwalter.
dump	Pseudofilesystem für interne Systemdienste.

Die folgenden Filesysteme enthalten die Benutzerkataloge:

tmp	/tmp: Katalog mit nur temporär angelegten Dateien für alle Benutzer; die Dateien werden nach Ende des Jobs gelöscht. Hier können auch "große" Dateien erzeugt werden; wenn im Einzelfall nichts anderes vereinbart ist, darf ein Benutzer bis zu 1 GByte oder bis zu 500 Dateien (*inodes*) anlegen. Er sollte "seinen" Katalog nur über die Systemvariable $TMPDIR ansprechen.

Verteilung der Filesysteme auf die Geräte des ZIB



home — /home: Katalog für alle Benutzer mit längerfristig angelegten kleineren Dateien, die vom ZIB nicht auf Magnetband gesichert werden; insbesondere sollten hier benutzerspezifische Vereinbarungsdateien liegen (z.B. *.profile, .netrc, .alhost* etc.); jeder Benutzer darf bis zu 4 MByte Daten oder bis zu 200 Dateien (*inodes*) in *home* abspeichern. Der Benutzer sollte "seinen" Katalog nur über die Variable $HOME ansprechen.

Die folgenden Filesysteme liegen auf Platten der UNIX-Workstation *serv01* (siehe Kapitel 1.3.4) und können sowohl von der CRAY im Batch als auch von der UNIX-Workstation *ufer* (Akronym für UNIX front-end relay) im Dialog oder über Dateitransfer mit FTP (siehe Kapitel 6) angesprochen werden. Sie enthalten langfristig angelegte Dateien, die vom ZIB regelmäßig auf Magnetband gesichert werden. Jeder Benutzer darf, sofern im Einzelfall nichts anderes vereinbart wurde, bis zu 20 MByte Daten abspeichern. Der Benutzer sollte "seinen" Katalog nur über die Variable $PERM ansprechen, da bei Vergrößerung der Plattenkapazität neue Namen von Filesystemen gebildet werden müssen. Ein Zugriff über die Systemvariable $PERM ist beim Dateitransfer-Kommando *ftp* jedoch nicht möglich.

bf — /bf: Katalog mit langfristig angelegten Dateien für Benutzer aus der FU Berlin

bt — /bt: Katalog mit langfristig angelegten Dateien für Benutzer aus der TU Berlin

bz — /bz: Katalog mit langfristig angelegten Dateien für Benutzer aus BAM, HMI und ZIB

n — /n: Katalog mit langfristig angelegten Dateien für Benutzer aus Niedersachsen

s — /s: Katalog mit langfristig angelegten Dateien für Benutzer aus Schleswig-Holstein

Greift man im CRAY-Job auf Dateien dieser Filesysteme zu, so muß der CRAY-Job die QSUB-Anweisung

```
#QSUB -ZIB S1
```

enthalten (siehe Kapitel 3.2, Parameter -ZIB).

Die folgenden Begriffe kennzeichnen Bereiche im SSD, sind jedoch keine Filesysteme:

ldcache — *logical device cache:* dieser Bereich im SSD (840 MByte) wird logisch unterteilt, den Filesystemen *tmp* und *home* wird je ein Teil des *ldcache* zur Verfügung gestellt. Neu angelegte Dateien werden zunächst im *ldcache* abgelegt, ebenso werden Dateien (bzw. Teile davon), die bearbeitet werden, vom System in den *ldcache* gelegt. Der Benutzer arbeitet tatsächlich in den meisten Fällen daher nicht mit Dateien auf der Magnetplatte, sondern mit den Informationen, die im SSD liegen, ohne selbst am eigenen Programm dafür etwas getan zu haben. Das System sorgt für gelegentliches Herausschreiben des *ldcache* auf die Magnetplatten und lädt Dateien in den *ldcache*, wenn sie benötigt werden. Die Erfahrung zeigt, daß in *tmp* weit über 99% aller Dateizugriffe über den *ldcache* abgewickelt werden können.

SDS — *secondary data segments:* Dieser Bereich im SSD (88 MByte) dient dazu, temporäre Dateien mit extrem großen Zugriffsraten während der Ausführung von Programmen zu verwenden. Dieser Teil des SSD steht dem Benutzer während dieser Zeit exclusiv zur Verfügung, er muß daher über NQS-Anweisungen (vgl. Kap. 3.2; Parameter -lQ) angefordert werden. Jobs mit SSD-Anforderungen werden in speziellen Jobklassen abgearbeitet.

Überwachung der Quotierung von Dateien

Dateien in den Filesystemen *home* und *tmp* werden auf der CRAY durch ein Quotierungssystem überwacht. Für jeden Benutzer sind Maximalwerte für die Datenmenge und für die Anzahl an Dateien eingetragen.

home: 4 MByte, 200 Dateien
tmp : 1 GByte, 500 Dateien

Zur Zeit werden bei Überschreitung dieser Werte nur Warnungen ausgegeben; zukünftig gilt:

Beim Überschreiten eines dieser Werte bricht der Job ab, und der Benutzer muß (für das Filesystem *home*) in einem separaten Job den Katalog aufräumen.

Für Quoting-Informationen über die Filesysteme *home* und *tmp* enthält der ZIB-Standard-Prolog *$HOME/.profile* (siehe Kapitel 3) sowohl einen Aufruf des informierenden Prozesses (quotamon) als auch am Anfang und Ende des Jobs das Kommando *quota*, mit dem der aktuelle Stand der Belegung benutzerspezifisch ausgegeben wird. Zur Zeit wird eine Warnung ausgegeben, wenn die aktuellen Werte mindestens 80% der vorgegebenen Grenzen für die Datenmenge (Size) und/oder Anzahl der Dateien (Inodes) betragen. Die von *quota* in der Spalte *Size* ausgegebene Datenmenge beschreibt den physikalisch von Dateien des Benutzers belegten Plattenspeicher, wobei die Einheiten, in denen Speicher zugewiesen wird, von der Konfiguration des gesamten Hintergrundspeichers abhängig sind. Daraus können sich starke Abweichungen zu Werten ergeben, wie sie z.B. von Kommandos wie *du* oder *ls* geliefert werden. Bei der Zählung der *Inodes* werden nur normale Dateien und Dateikataloge (aber z.B. keine *links*, denn diese haben dieselbe *Inode*-Nummer) erfaßt. Da unter UNICOS im *Inode* neben Verwaltungsinformationen in geringem Umfang auch Dateiinhalte abgelegt werden, kann es im Extremfall vorkommen, daß zu einer Anzahl *Inodes* größer Null unter *Size* die Datenmenge Null ausgewiesen wird. Das Kommando *quota* kann an beliebiger Stelle im Job stehen und erlaubt über seinen Exitstatus eine Analyse des Zustands; u.a. gilt

Exitstatus	Bedeutung
0	alles in Ordnung
17	'warning level' erreicht
18	'limit level' erreicht

Mit Hilfe des Kommandos *du* (bzw. *du -s $HOME*) wird man im Unterschied zu *quota* über den tatsächlichen Umfang des Inhalts der angelegten Dateien eines Kataloges (in Blöcken) informiert (auf der CRAY entspricht 1 Block 4 KB, auf der SUN 0,5 KB).

Weitere Informationen erhält man durch das UNICOS-Kommando

```
man quota
```

Lebensdauer von Dateien

Dateien im Filesystem *tmp* werden am Ende jedes Jobs gelöscht. Da in besonderen Fällen dieses Löschen unterbleiben kann, z.B. bei speziellen Jobabstürzen, wird dieser Katalog vom ZIB zusätzlich regelmäßig aufgeräumt. Falls Sie im Katalog */tmp* Dateinamen oder Katalognamen verwenden, die nicht den Konventionen ($TMPDIR) entsprechen, müssen Sie damit rechnen, daß ggf. diese Dateien auch während des Programmlaufs gelöscht werden.

Dateien im Filesystem *home* existieren zunächst unbegrenzt. Beachten Sie, daß Dateien in */home* vom ZIB nicht auf Magnetband gesichert werden. Dateien, die für Sie wichtig sind, müssen Sie zu einem der Vorrechner (Datensicherung durch das ZIB) oder zu Ihrem eigenen Rechner (eigene Datensicherung) kopieren. Insofern endet die Lebensdauer der Dateien in *home* mit einer Neugenerierung dieses Filesystems, wie sie z.B. nach einem Hardwarefehler auf dem zugehörigen Plattenlaufwerk notwendig ist.

Dateien in den Filesystemen *bf*, *bt*, *bz*, *n* und *s* liegen auf der UNIX-Workstation *serv01* und enthalten langfristig angelegte Dateien, die vom ZIB regelmäßig auf Magnetband gesichert werden. Jeder Benutzer darf, sofern im Einzelfall nichts anderes vereinbart wurde, bis zu 20 MByte Daten abspeichern.

Festlegung der Zugriffsrechte im ZIB

Eine Benutzergruppe (group) im Sinne der Dateizugriffsrechte von UNIX (vgl. Kap. 2.3) ist zunächst der Benutzer selbst, d.h. jeder eingetragene Benutzer auf der CRAY X-MP des ZIB ist allein in einer Gruppe; diese Gruppe hat den gleichen Namen wie der Benutzer selbst. Verschiedene Benutzer, die z.B. an gleichen Projekten arbeiten, können auf schriftlichen Antrag hin, der mit der Unterschrift dieser Benutzer versehen ist (senden an: ZIB, Abt. Betrieb, z.Hd. Hrn. Götz, vgl. Kap. 1.7), eine Gruppe bilden; diese wird dann vom ZIB eingetragen.

2.8 Einige System-Kataloge

Im UNICOS-Dateibaum sind immer einige spezielle Kataloge vorhanden, die in erster Linie systemeigene Dateien enthalten. Die wichtigsten sind:

/bin	*binaries*; in diesen Katalogen liegen die wichtigsten Dienstprogramme (z.B. liegen *ls* und *pwd* in */bin*; die seltener benutzten Dienstprogramme liegen in */usr/bin*)
/sbin	Kopie von */bin* im *SSD*
/dev	*devices*; in diesem Katalog liegen alle Geräteeinträge (special files).
/etc	*etcetera*; in diesem Katalog liegen Systemverwaltungsprogramme und die meisten Systeminformationsdateien (z.B. */etc/passwd*).
/tmp	*temporary*; dieser Katalog wird vom System in regelmäßigen Abständen gelöscht. Er kann von jedem Benutzer und Benutzerprogrammen als temporärer Speicherbereich benutzt werden (er sollte nur über die Variable $TMPDIR verwendet werden (vergl. Kapitel 2.7)).
/lib	*libraries*; dieser Katalog enthält einen Teil der Systembibliotheken.
/usr	in diesen Katalog werden oft weitere Dateisysteme eingehängt. Außerdem enthält dieser Katalog noch etliche weitere System-Kataloge:

/usr/adm	in diesem Unterkatalog werden Informationen zur Systemverwaltung abgelegt.
/usr/bin	siehe */bin*.
/usr/include	in diesem Unterkatalog liegen die sogenannten Headerdateien, die zur Programmentwicklung mit C benutzt werden.

/usr/lbin	in diesem Unterkatalog liegen Programme und Prozeduren, die vom ZIB bereitgestellt wurden.
/usr/lib	in diesem Unterkatalog liegen weitere Systembibliotheken sowie die vom ZIB und anderen Rechenzentren bereitgestellten Bibliotheken (z.B. nag, grips etc.)
/usr/man	in diesem Katalog liegt die Online-Dokumentation des UNICOS-Systems.
/usr/spool	dies ist ein Sammelkatalog für das UNICOS-Spooling-System.
/usr/ucb	in diesem Katalog stehen Dienstprogramme, die nur bei der UNIX-Version BSD 4.2 (Berkley-UNIX) vorkommen.

3. Ausführung eines UNICOS-Batchjobs

Ein UNICOS-Batchjob ist, vereinfacht dargestellt, eine Folge von UNIX- (bzw. UNICOS-) Anweisungen. Die Anweisungen, die der Anwender sonst in einer UNIX-Dialogsitzung nacheinander eingibt, werden zeilenweise in einer Datei (in der Regel auf einem Vorrechner) abgelegt. Dazu gehören auch die Zeilen, die von der Standardeingabedatei in eine andere Datei umgelenkt werden (siehe Kapitel 4.4.1).

3.1 Ein einfaches Beispiel

Ein UNICOS-Batchjob wird auf einem beliebigen Rechner außerhalb der CRAY erzeugt; der Benutzer verwendet dafür die ihm bekannten Hilfsmittel auf diesem Rechner, insbesondere einen ihm vertrauten Editor. Am Anfang des Jobs müssen NQS-Anweisungen stehen; diese werden im Kapitel 3.2 erläutert. NQS ist die Abkürzung für *Network Queueing System* und stellt die Umgebung dar, mit der man mit dem vornehmlich für Dialoganwendungen konzipierten System UNIX Batchanwendungen betreiben kann. Im Norddeutschen Vektorrechnerverbund wurde NQS bislang nur auf den UNICOS-Systemen implementiert.

```
# USER=userid PW=password     # NQS-Anweisung: Benutzervalidierung
# QSUB-r first              # NQS-Anweisung: Festlegung des Jobnamens
# QSUB-lm 1Mw               # NQS-Anweisung: Hauptspeicheranforderung
                            # (1 Mword)
# QSUB-lT 2:30              # NQS-Anweisung: Rechenzeitanforderung (2
                            # Minuten, 30 Sekunden)
# QSUB -ZIB none            # NQS-Anweisung: kein Dateitransfer vom/zum
                            # Vorrechner
                            # => Ende des NQS-Teils <=
cd $TMPDIR                  # Wechsel in den temporären Katalog
cat > first.f << '%%'       # Erzeugen der Datei first.f
      write (*,*) ' hello world'
      end
%%
cft77 -es first.f           # Aufruf des Fortran-Compilers
cat first.l                 # Kopieren des Compilerlistings nach stdout
segldr first.o              # Aufruf des Segmentladers
a.out                       # Ausführen des Fortran-Programms
exit                        # Ende des Jobs
```

Der so auf einem beliebigen Rechner erzeugte UNICOS-Batchjob wird (sofern nicht schon auf einem Vorrechner erzeugt) zu einem der CRAY-Vorrechner gebracht (siehe Kapitel 5) und dann mit Hilfe des SUBMIT-Kommandos der CRAY Station Software (siehe Kapitel 5.1) oder mit Hilfe des ALRJE-Dienstes (siehe Kapitel 6) zur CRAY übertragen und dort ausgeführt. Man beachte, daß *-lm* (kleines m!) und *-lT* (großes T!) für die Berechnung der Priorität zur Jobausführung im ZIB verwendet werden (siehe Kapitel 3.3); für *-lM* und *-lt* werden, falls nicht vom Benutzer angegeben, Standardwerte verwendet (siehe Kapitel 3.23).

Nach Abarbeitung des Jobs wird die Standardfehlerdatei *stderr*, ggf. ergänzt um die Standardausgabe *stdout*, als Ausgabe an den Vorrechner zurückgeschickt.

3.2 Network Queueing System (NQS)

Da die NQS-Anweisungen gemeinsam mit den Anweisungen für die UNIX-Shell in einer Datei stehen, müssen diese durch # (hash-mark, Doppelkreuz) in Spalte 1 für die Shell als Kommentar gekennzeichnet sein. Die erste Anweisung dient stets der Benutzervalidierung, sie hat folgendes Format:

```
# USER=userid PW=password
```

Die Benutzeridentifikation *userid* besteht im ZIB aus genau acht Zeichen, dabei sind nur Kleinbuchstaben und Ziffern zugelassen. Das erste Zeichen ist ein Buchstabe, er repräsentiert das Bundesland des Benutzers:

b - Berlin
n - Niedersachsen
s - Schleswig-Holstein

Das zweite Zeichen kennzeichnet in Verbindung mit dem ersten Zeichen die Organisation (in der Regel die Universität), über die der Benutzer seinen Antrag auf Zulassung zur CRAY gestellt hat:

bf - Freie Universität Berlin
bt - Technische Universität Berlin
bz - Konrad-Zuse-Zentrum für Informationstechnik Berlin

na - Technische Universität Braunschweig
nb - Technische Universität Clausthal
nc - Gesellschaft für Wissenschaftliche Datenverarbeitung Göttingen
nh - Regionales Rechenzentrum für Niedersachsen an der Universität Hannover
nj - Universität Osnabrück

s - Universität Kiel

Die übrigen Zeichen der Benutzeridentifikation werden nach Kriterien des lokalen Rechenzentrums vergeben. Arbeitet der Benutzer auch mit der CYBER 930 (NOS/VE) des ZIB, so gilt für diesen Rechner dieselbe Benutzeridentifikation. Dieselbe Benutzeridentifikation gilt auch für die CRAY der Universität Kiel, sofern der Benutzer auch auf diesem Rechner zugelassen ist. Die übrigen NQS-Anweisungen dienen der Festlegung von Eigenschaften des UNICOS-Batchjobs:

QSUB option *wert* option *wert* ... option *wert*

-eo Die Standardfehlerdatei *stderr* wird in die Standardausgabedatei *stdout* kopiert.
Achtung: Auf Grund unterschiedlicher Puffer erfolgt die Ausgabe beliebig gemischt und nicht unbedingt chronologisch; sollte bei Verwendung des ZIB-Standard-Profile (siehe Kapitel 3.5) nicht gesetzt werden!

-lf *limit maximum process file space*: Maximale Dateigröße jedes einzelnen Prozesses; keine Angabe von *-lf* bedeutet *-lf 200 Mb*; maximal sind *-lf 2,5 Gb* möglich (die NQS-Kommandosyntax verlangt hier, entgegen der allgemein Schreibweise, ein kleines *b* für die Angabe in Byte!). Dieses Limit beeinflußt die Einteilung in die jeweilige Jobklasse.

-lM *limit maximum memory per job:* Maximale Hauptspeicheranforderung für den Job; die Angabe kann in folgenden Einheiten erfolgen: *b* (bytes) oder *w* (words), jeweils ergänzt um den Faktor *K* (Kilo = 1024), *M* (Mega = 1024*1024) oder *G* (Giga = 1024*1024*1024). Keine Angabe von *-lM* wirkt wie *-lM 2 Mw*, falls *-lT* kleiner gleich 20 oder nicht angegeben; sonst *-lM 6 Mw*; maximal sind *-lM 6 Mw* möglich.

Beispiel:

```
-lM 1Mw
```

-lm *limit maximum memory per process:* Maximale Hauptspeicheranforderung für den Prozeß; die Angabe erfolgt wie bei *-lM*. Keine Angabe von *-lm* wirkt wie *-lm 1 Mw*, falls *-lT* kleiner gleich 20 oder nicht angegeben; sonst *-lm 3 Mw*; maximal sind *-lm 3 Mw* möglich. Dieses Limit geht in die Berechnung der Ressourcen-Priorität RP (siehe Kapitel 3.3) mit ein.

-lQ *limit maximum secondary memory:* Maximale Anforderung für den Hintergrundspeicher (Secondary Data Storage - SDS), Teil des SSD; Angabe des Limits wie *-lM*; keine Angabe von *-lQ* bedeutet kein SDS; maximal sind zur Zeit *-lQ 88 Mb* möglich. Dieses Limit beeinflußt die Einteilung in die jeweilige Jobklasse.

-lT *limit maximum CPU time per job:* Maximale CPU-Zeit aller Prozesse des Batchjobs, in [[stunden:]minuten:]sekunden; die Minuten und Sekunden können auch größer als 60 sein; keine Angabe von *-lT* bedeutet *-lT 5*, falls *-lm* kleiner gleich *1 Mw* oder nicht angegeben; sonst *-lT 20000* Sekunden; maximal sind *-lT 20000* Sekunden möglich. Dieses Limit geht in die Berechnung der Ressourcen-Priorität RP (siehe Kapitel 3.3) mit ein.

Beispiel:

```
-lT 2:30
```

-lt *limit maximum CPU time per process:* Maximale CPU-Zeit jedes einzelnen Prozesses des Batchjobs, in [[stunden:]minuten:]sekunden; die Minuten und Sekunden können auch größer als 60 sein; keine Angabe von *-lt* bedeutet *-lt 5*, falls *-lm* kleiner gleich *1 Mw* oder nicht angegeben; sonst *-lt 20000* Sekunden; maximal sind *-lt 20000* Sekunden möglich.

-q qname *qname*: Der Job soll ausdrücklich in der genannten Klasse *qname* ablaufen. Die Limits des Jobs dürfen die entsprechenden Werte der angegebenen Klasse nicht überschreiten (siehe Kapitel 3.3). Jobs, die auf einem Vorrechner ein Magnetband benötigen, müssen z.B. in der Klasse *magtape* ablaufen.

-r *request-name*: Name des Batchjobs; keine Angabe von -r setzt die Benutzeridentifikation als Namen des Batchjobs ein. Bei Zugang über die NOS/BE-Station ist dieser Parameter wirkungslos.

-ZIB *ZIB*: Es besteht die Möglichkeit, mit Hilfe dieser QSUB-Anweisung zu erklären, welche Vorrechner der Job für Dateitransfers während seiner Ausführung benötigt. Bis zu zwei Angaben *VE*, *BE*, *S1* oder *S2* können beliebig verknüpft werden (z.B. VEBE oder BES1 usw.;). Wird keine Angabe zu den benötigten Vorrechnern gemacht, wird als Voreinstellung der Vorrechner genommen, über den der Job in die CRAY gelangt ist. Erlaubt sind folgende Angaben:

all	Der Job benötigt alle angeschlossenen Vorrechner.
S1	Der Job benötigt die SUN-Anlage *serv01* des ZIB (auch bei Verwendung von FTP, ALRJE, WO2BE und Dateien in den über $PERM erreichbaren Filesystemen).
S2	Der Job benötigt die SUN-Anlage *serv02* des ZIB.
VE	Der Job benötigt die NOS/VE-Anlage des ZIB (auch bei Dateitransfer mit *fetch*, *dispose* und *acquire* über das CDCNET).
BE	Der Job benötigt die NOS/BE-Anlage des ZIB (auch bei *getbe* oder *putbe*).
none	Der Job benötigt keinen der angeschlossenen Vorrechner.

3.3 Jobklassen und Prioritäten

3.3.1 Jobklassen

Jeder Job, der in die *Input Queue* der CRAY gelangt, wird nach den Anforderungen von Betriebsmitteln in den QSUB-Anweisungen des *NQS* (CPU-Zeit *-lt*, Hauptspeicheranforderung für den einzelnen Prozeß *-lm*, Magnetbandbedarf *-q*, SDS-Bedarf *-lQ*) einer bestimmten Jobklasse zugeordnet.

Für kurze Entwicklungsjobs gilt:

Name der Klasse:	CPU in Sek.	Memory KWorte, dez.	Magn. Tape	SDS	Anzahl Jobs: User	gesamt
express	< = 5	< = 1000	-	-	1	4
kurz	< = 20	< = 1000	-	-	1	4
magtape	< = 20	< = 1000	1	-	1	4

Ein Job der Klasse *kurz* oder *express* wird vom *Scheduler* aus der *Input Queue* in die *Execution Queue* übernommen, sobald seine NQS-Nummer die kleinste in seiner Klasse ist (first in - first out), die Maximalzahl aktiver Jobs in dieser Klasse noch nicht erreicht ist oder unterschritten wird und der Benutzer nicht schon einen Job in einer der Klassen *express* oder *kurz* in Ausführung hat. In der Regel wird der Job unmittelbar nach Einbringen in die *Input Queue* zur Ausführung gebracht. Diese beiden Klassen sind für Jobketten **nicht** zulässig. Jobketten in diesen beiden Klassen behindern alle Benutzer erheblich; das ZIB wird solche Jobketten abbrechen.

Jobs, die auf einem der Vorrechner ein Magnetband benötigen, müssen in den NQS-Anweisungen zusätzlich die Angabe

```
QSUB -q magtape
```

aufführen, damit der Job in der Klasse *magtape* abgearbeitet wird.

Alle übrigen Jobs, also lange Jobs oder Produktionsjobs, werden entweder in der Klasse *warte* abgelegt oder, falls Dateien verarbeitet werden, die größer als 200 MByte sind, entweder in *bigfile* (200 MByte < = Dateigröße < = 600 MByte) oder in *hugfile* (600 MByte < = Dateigröße < = 2,5 GByte). Diese Klassen bilden eine *pre input queue*, d.h., diese Jobs werden nicht aus diesen Klassen heraus zur Ausführung gebracht. Auf der CRAY läuft alle 5 Minuten ein Überwachungsprogramm ab, welches die Jobs, die zur Ausführung gebracht werden sollen, nach geeigneten Kriterien in eine der Jobklassen *tags*, *nachts* oder *wochend* der eigentlichen *input queue* eingeordnet. Als Kriterium dient die Priorität *P* (= RP + AP, siehe Kapitel 3.3.2), wobei berücksichtigt wird, daß sich pro Benutzer nicht mehr als ein Job sowie insgesamt nicht mehr als ein Job mit großen Dateien *bigfile* oder *hugfile* in der Ausführung befinden. In der Regel werden Jobs in den Klassen *tags*, *nachts* und *wochend* umgehend zur Ausführung gebracht.

Jobs in der Klasse *hugfile* werden nach Rücksprache mit dem ZIB nur zu speziellen Zeiten "per Hand" zur Ausführung gebracht. Läuft ein Job der Klasse *hugfile*, können nur noch Jobs der Klasse *smafile* zusätzlich laufen. Die Klasse *smafile* entspricht der Klasse *warte* mit einer maximalen Dateigröße von 10 MByte je Prozeß. Der Benutzer muß seinen Job explizit durch

```
#QSUB -q smafile
#QSUB -lf 10Mb
```

der Klasse *smafile* zuordnen (die NQS-Kommandosyntax verlangt hier, entgegen der üblichen Schreibweise, ein kleines *b* für die Angabe in *Byte*!).

Jobs, die die zugelassenen Limits überschreiten, fallen in die Klasse *garbage*. Diese Jobs kommen nicht zur Bearbeitung; Jobs in dieser Klasse werden regelmäßig entfernt. Daher ergeben sich folgende zulässige Maximalwerte für die einzelnen Limits der QSUB-Anweisungen:

Maximal- und Standardwerte für Joblimits:

			Maximum:	Default:
CPU-Zeit -				
- pro Prozeß	t	<=	20000 [Sekunden]	5 [Sekunden]
- pro Job	T	<=	20000 [Sekunden]	5 [Sekunden]
Memory -				
- pro Prozeß	m	<=	3 Mw	1 Mw
- pro Job	M	<=	6 Mw	2 Mw
max. Dateigröße -				
- pro Prozeß	f	<=	2.5 Gb	200 Mb
SDS-Limit	Q	<=	88 Mb	0 Mb

(in der Tabelle sind die Werte in der NQS-Kommandosyntax angegeben, die, entgegen der üblichen Schreibweise, ein kleines *b* für die Angabe in *Byte* und einen Dezimalpunkt anstelle eines Dezimalkommas verlangt!). Für Magnetbandjobs gilt:

```
t   <=   20 [Sekunden] und m <= 1000 Kw
```

Sollten Ihre Jobs innerhalb dieser Maximalwerte nicht ausführbar sein, so bitten wir um Rücksprache.

Um die große Anzahl von CRAY-Jobs auf einem praktikablen Maß zu halten und für einzelne Jobs akzeptable Bearbeitungszeiten zu erreichen, dürfen pro Auftragsnummer nicht mehr als sechs Jobs in der CRAY (INPUT und EXECUTION Queue) abgelegt sein. Überzählige Jobs gelangen in die Klasse *toomany*, in der sie nicht zur Abarbeitung kommen; gibt der Benutzer mehr als 10 Jobs ab, so werden vom siebten Job an alle sofort gelöscht.

3.3.2 Prioritäten

Auf der CRAY läuft alle 5 Minuten ein Überwachungsprogramm ab, welches die Jobs, die zur Ausführung gebracht werden sollen, nach geeigneten Kriterien in eine der Jobklassen *tags*, *nachts* oder *wochend* der eigentlichen *Input Queue* zugeordnet (siehe Kapitel 3.3.1). Dieses Kriterium ist die Priorität *P* (=RP+AP), die sich aus der Ressourcen-Priorität RP und der Alterungspriorität AP zusammensetzt. Das Überwachungs-Programm hat folgende Leistungen:

* Berechnen der Ressourcen-Priorität RP für jeden Job.
* Erhöhen der Alterungspriorität AP **eines** Jobs jedes Benutzers.
* In Abhängigkeit von der freien Kapazität der *Execution Queue* (Zahl der Jobs, Summe der angeforderten Feldlänge aller aktiven Jobs) werden eine gewisse Zahl von Jobs entsprechend der Betriebszeit in eine der Jobklassen *tags*, *nachts* oder *wochend* der eigentlichen *Input Queue* zugeordnet. Die Auswahl erfolgt nach der Priorität *P* (=RP+AP), wobei berücksichtigt wird, daß sich pro Benutzer nicht mehr als ein Job sowie insgesamt nicht mehr als ein Job mit großen Dateien *bigfile* in der Ausführung befinden.
* Aufbereiten von Informationen über die Queues einschließlich der Werte der Prioritäten, Ablage auf der CRAY, CYBER 825 (NOS/BE), CYBER 930 (NOS/VE), UNIX-Vorrechner *ufer* sowie den CD-Anlagen von FUB und TUB, so daß der Benutzer jeweils die Information mit dem *Remote Queue Status Programm* RST (siehe Kapitel 5.6) abrufen kann.

Die Ressourcen-Priorität RP

Die Ressourcen-Priorität RP berücksichtigt die angeforderten Ressourcen des Jobs ; z. Zt. sind dies die geforderte CPU-Zeit (T-Parameter) und der geforderte Hauptspeicher (m-Parameter in Kilo-Worten).

Diese Priorität berechnet sich wie folgt:

$$RP' = 100 - 100 * LOG\left(\frac{t}{50000 - t}\right) - 70 * LOG\left(\frac{m}{4000 - m}\right)$$

LOG: natürlicher Logarithmus

```
RP =    0    (für RP' <= 0)
RP =  RP'    (für 0 <= RP' <= 1000)
RP = 1000    (für RP' > 1000)
```

Einen Überblick über die Werte für RP der CRAY X-MP liefert die folgende Tabelle:

m \ t	4	10	40	100	400	1000	4000	10000	20000
100000	1000	1000	1000	977	838	745	600	494	369
200000	1000	1000	1000	927	788	695	550	444	346
300000	1000	1000	988	896	757	664	519	413	316
400000	1000	1000	966	874	735	642	497	391	294
500000	1000	1000	949	857	718	625	480	374	276
600000	1000	1000	934	842	703	610	465	359	261
700000	1000	1000	921	829	690	597	452	346	249
800000	1000	1000	910	818	679	586	441	335	237
900000	1000	1000	899	807	668	575	430	324	227
1000000	1000	1000	889	797	658	565	420	314	217
1200000	1000	1000	872	780	641	548	403	297	199
1400000	1000	994	856	764	625	532	387	281	183
1600000	1000	979	841	749	610	517	372	266	168
1800000	1000	965	827	735	596	503	358	252	154
2000000	1000	951	813	721	582	489	344	238	140
2200000	1000	937	799	707	567	474	329	223	126
2400000	1000	923	785	692	553	460	315	209	112
2600000	1000	908	770	677	538	445	300	194	97
2800000	984	892	753	661	522	429	284	178	81
3000000	966	874	736	644	505	412	267	161	63

Auf den Vorrechnern des ZIB steht das Programm *PRIOCR* zur Verfügung, mit dem man sich -auch ohne Taschenrechner- Werte für *RP* in Abhängigkeit von *T* und *m* ausrechnen lassen kann. (Sinnvoll ist der interaktive Aufruf von *PRIOCR*.)

Die Alterungspriorität AP

Jeder neu in die *Input Queue* aufgenommene Job erhält die Alterungspriorität AP = 0. Bei jedem Überwachungslauf (alle 5 Minuten) wird bei jedem Benutzer die Alterungspriorität seines Jobs mit kleinster *NQS*-Nummer (d.h. der Älteste) um 1 bis zu einem Maximalwert von 8000 erhöht. Der Wert von 8000 wird frühestens nach 32 Tagen ereicht. Hat der erste Job die Alterungspriorität 1000 erreicht, so kann ein zweiter Job dieses Benutzers ebenfalls altern; hat dieser 1000 erreicht, so kann ein dritter Job altern etc.

Der ZIB-Prioritätenalgorithmus läßt normalerweise kleine Jobs früher abarbeiten als große; hierdurch soll jedoch ein Benutzer, der einen großen Job in viele kleine zerlegt, nicht bevorzugt werden. Daher wird für jeden ausgeführten Job einer Benutzerkennung ein Malus vermerkt, der sich aus der Anzahl der Jobs dieser Benutzerkennung an diesem Tag seit 0.00 Uhr, multipliziert mit einem Grundwert (z.Zt. 4) ergibt. Dieser Malus wird von der Alterspriorität *AP* der neu hinzugekommenen Jobs subtrahiert, wodurch diese Jobs nicht mit Null, sondern mit einer negativen Alterungspriorität zu altern anfangen.

3.3.3 Abarbeiten der Jobs zu speziellen Betriebszeiten

Es werden die drei Betriebszeiten Tages-, Nacht- und Wochenend-Betrieb sowie die drei Jobklassen *tags*, *nachts* und *wochend* unterschieden.

Betriebszeiten:

Tagesbetrieb:	montags - freitags jeweils 7^{00} bis 17^{00} Uhr
Nachtbetrieb:	montags 0^{00} bis 7^{00} Uhr und
	montags - donnerstags jeweils ab 17^{00} Uhr bis 7^{00} Uhr des darauffolgenden Tages
Wochenendbetrieb:	freitags 17^{00} Uhr bis sonntags 24^{00} Uhr

Klassen:

In Abhängingkeit von der Ressourcen-Priorität RP ist jeder Job einer dieser drei Klassen zugeordnet und kann zu folgenden Betriebszeiten gestartetet werden:

Klasse	Ressourcen-Priorität	Betriebszeit
tags	RP > = 600	Tages-, Nacht- oder Wochenendbetrieb
nachts	300 < = RP < 600	Nacht- oder Wochenendbetrieb
wochend	RP < 300	Wochenendbetrieb

3.4 Standards für die Druckaufbereitung

Unter UNICOS treffen zwei Standards für die Druckaufbereitung zusammen, die "klassische" mit den Vorschubsteuerzeichen " " (Leerzeichen), 0, 1 und + in Spalte 1 jeder Zeile und Beginn des Ausdrucks ab Spalte 2, im folgenden mit ASA-Steuerzeichen bezeichnet, und die "moderne" mit den ASCII-Steuerzeichen *form feed* (ASCII-Code 12) und *horizontal tabulator* (ASCII-Code 9) und Beginn des Ausdrucks ab Spalte 1, im folgenden mit ASCII-Steuerzeichen bezeichnet.

ASA-Steuerzeichen werden von folgenden Produkten erzeugt bzw. erwartet:

* FORTRAN-Compiler CFT und CFT77
* (alte) FORTRAN-Programme
* Drucker an der CYBER 825 unter NOS/BE
* Drucker an der CYBER 930 unter NOS/VE, sofern bei dieser Datei FILE_CONTENTS = LIST gesetzt ist
* Ausgabedateien, die mit dem RJE-Dienst des DFN von NOS/VE aus oder von NOS/BE aus verschickt werden sollen

ASCII-Steuerzeichen werden von folgenden Produkten erzeugt bzw. erwartet:

* sämtliche UNICOS-Kommandos (segldr, ls, ja, dbx etc.)
* PASCAL-Compiler
* C-Compiler
* (neue) PASCAL- und C-Programme

* Drucker an der CYBER 930 unter NOS/VE, sofern bei dieser Datei *FILE_CONTENTS = LEGIBLE* (oder *= UNKNOWN*) gesetzt ist, bei Verwendung von *dispose -d PR* wird *LEGIBLE* standardmäßig gesetzt.
* Drucker an einem UNIX-Rechner

Ein kompletter Job mit FORTRAN-Liste und UNICOS-Kommando-Protokollierung enthält also sowohl Daten mit ASA-Steuerzeichen (FORTRAN-Programm) als auch mit ASCII-Steuerzeichen (Kommando-Protokollierung). Sofern man die Ausgabedatei *stdout* sich nur an einem Vorrechner im ZIB ansieht, ist dies unschädlich. Auch Drucken auf einem NOS/VE-Drucker (FC = LEGIBLE) oder UNIX-Drucker ist möglich, jedoch werden die ASA-Steuerzeichen nicht als Steuerzeichen interpretiert, sondern in Spalte 1 gedruckt. Man darf eine solche Datei jedoch nicht mit dem RJE-Dienst des DFN verschicken, da hier in jedem Fall (auch bei NOS/VE FC = LEGIBLE!) ASA-Steuerzeichen erwartet werden und die Angaben in Spalte 1 als Steuerzeichen interpretiert werden und, wenn sie von den erlaubten Steuerzeichen abweichen, nicht übertragen werden.

UNICOS bietet zwei Programme an, um eine Datei mit Steuerzeichen des einen Typs in eine des anderen Typs umzuwandeln:

```
asa file
```

Das Kommando *asa* interpretiert die erste Spalte von *file* als ASA-Steuerzeichen und setzt sie um in ASCII-Steuerzeichen. Die Ausgabe erfolgt nach *stdout*.

```
nasa file
```

Das Kommando *nasa* nimmt als Eingabe *file* und setzt ASCII-Steuerzeichen um in ASA-Steuerzeichen, der Text wird um eine Spalte nach rechts verschoben. Die Ausgabe erfolgt nach *stdout*.

Beispiel (vgl. Beispiel aus Kap. 3.1):

```
...
cft77 -es first.f # Aufruf des Fortran-Compilers
asa first.l       # Kopiere Compilerlisting mit
                  # ASCII-Steuerzeichen nach stdout
...
```

Da die Mehrzahl der Benutzer des ZIB über DFN-Dienste mit der CRAY arbeitet, wird durch Anweisungen in der Datei *.profile*, die als Standardprolog jedem Benutzer vom ZIB angelegt wurde (Kap. 3.5), erreicht, daß die Dateien *stdout* und *stderr* je Zeile um ein Zeichen nach rechts verschoben werden. Druckausgaben, die bereits im ASA-Format vorliegen, verlieren dadurch ihre Formatierung. Die Datei $STDASA wird ohne Zeichenverschiebung kopiert; ASA-Dateien in $STDASA behalten also ihre Vorschubsteuereung.

Beispiel:

```
cft -es -l$STDASA first.f
```

3.5 ZIB-Standard-Prolog

Das ZIB richtet jedem Benutzer in seinem Katalog $HOME eine Datei *.profile* ein, die am Anfang jedes Jobs durchlaufen wird. Jeder Benutzer kann diesen Prolog nach eigenen Wünschen ändern, jedoch sollte man bei Änderungen besonders vorsichtig verfahren.

Im ZIB-Standard-Prolog werden für einen Batchjob folgende Anweisungen ausgeführt:

* Feststellen des permanenten Kataloges $PERM (Dateien auf der UNIX Workstation *ufer*).
* Start des Filequoten-Informationsdienstes *quotamon* im Hintergrund.

* Feststellen der Jobherkunft ($ZIB_ORIGIN).
* Feststellen einiger Stationparameter (wenn der Job über eine Station kam: $ZIB_MF, $ZIB_TID, $ZIB_JSQ).
* Ausgabe von Informationen über den belegten Plattenplatz in den Katalogen $HOME und $TMPDIR.
* Erzeugen von Dateien *stdout*, *stderr* und *jobacct* im temporären Katalog $TMPDIR.
* Ausgabe des Gültigkeitszeitraumes des Jobpaßwortes durch das Script *pwval*.
* Ausdruck des ZIB-Headers mit aktuellen Informationen (falls die aktuelle Information vom Benutzer noch nicht abgerufen wurde).
* Ausgabe der unter der eigenen Benutzerkennung angesammelten Postinformation (mail -p).
* Ausführen des eigentlichen Jobs (*sh -ex*) mit Ausgabe auf temporäre Dateien, die für *stdout* bzw. *stderr* eingerichtet worden sind.
* Kopieren des temporären *stderr* (mit ASA-Steuerzeichen) nach *stdout*.
* Kopieren des temporären *stdout* (mit ASA-Steuerzeichen) nach *stdout*.
* Ausgabe einer ausführlichen Job- und Kommandostatistik.
* Kopieren der temporären, bereits mit ASA-Steuerzeichen versehenen Datei $STDASA nach *stdout*.

Dieser Prolog sorgt u.a. dafür, daß auch bei Verwendung des RJE-Dienstes des DFN Ausgabedateien korrekt übertragen werden; allerdings werden in der Datei *stdout* vorhandene ASA-Steuerzeichen (z.B. bei FORTRAN-Compilerlisten oder bei FORTRAN-Objektprogrammen) ebenfalls nach rechts verschoben und dadurch gedruckt anstatt interpertiert. In der Datei $STDASA hingegen werden diese Vorschübe korrekt interpretiert.

Die Anweisung *rm -rf $TMPDIR* darf im Job selbst nicht stehen, da man sonst die temporären Dateien *stdout* und *stderr* löscht und **überhaupt keine Ausgabe** erhält.

Der Inhalt des ZIB-Standard-Prolog *.profile*:

```
#------------------------------------------------------------------------
#
#    Default Bourne shell profile at ZIB (Version 1.5, 22.3.90)
#
#------------------------------------------------------------------------
# --->    Common for interactive and batch
#
#------------------------------------------------------------------------
#
PERM=$HOME/$LOGNAME
if [ ` echo $LOGNAME | wc -c ` -eq 9 ]
  then
  if [ ` echo $LOGNAME | cut -c 1 ` = 'b' ]
    then
    PERM=/` echo $LOGNAME | cut -c 1-2 `/$LOGNAME
    else
    PERM=/` echo $LOGNAME | cut -c 1 `/$LOGNAME
  fi
fi
export PERM
quotamon &
#------------------------------------------------------------------------
# --->    Batch only
#    NOTE : Don't delete all $TMPDIR within batch request
#         or this script will not find any output
#------------------------------------------------------------------------
if [ x$ENVIRONMENT = xBATCH ]
  then
  STDOUT=$TMPDIR/.STDOUT
  export STDOUT
  STDERR=$TMPDIR/.STDERR
```

```
  export STDERR
  STDASA=$TMPDIR/.STDASA
  export STDASA
  JOBACCT=$TMPDIR/.JOBACCT
  export JOBACCT
  ALRJE=/tmp/alrje/$LOGNAME
  export ALRJE
  COUNT_R=`echo "$QSUB_REQID" | wc -c`
  COUNT_H=7
  COUNT=`expr $COUNT_R - $COUNT_H`
  COUNT_BLANK=`expr 18 - $COUNT`
  NQS_ID=`echo "$QSUB_REQID" | cut -c1-$COUNT`
  BLANK=`echo '..................' | cut -c1-$COUNT_BLANK`
  SEARCH="^$BLANK $NQS_ID "
  FOUND=`uscpstat | grep "$SEARCH" `
  if [ x"$FOUND" != x ]
    then
    ZIB_ORIGIN='USCP'
    ZIB_MF=`echo "$FOUND" | cut -c67-68`
    ZIB_TID=`echo "$FOUND" | cut -c70-77`
    ZIB_TID=`echo  $ZIB_TID`
    ZIB_JSQ=`echo "$FOUND" | cut -c9-13`
    ZIB_JSQ=`echo $ZIB_JSQ`
    export ZIB_ORIGIN ZIB_MF ZIB_TID ZIB_JSQ
  fi
  echo " "
  echo " ================================================================================"
  echo "     Starting Request $QSUB_REQNAME ($QSUB_REQID) : "`date`
  echo " "
  echo "             stderr (standard error file & shell log file)"
  echo " ================================================================================"
  echo " "
#-------------------------------------------------------------------------
# --->    What's about Your password and ZIB's news ?
#
  if [ x$ZIB_ORIGIN = xUSCP ]
    then
    pwval | cpsf
    echo " Quotas in /home and /tmp at the start:"
    quota | cpsf
    news 2>/dev/null | cpsf
    mail -p | cpsf
  else
    pwval
    echo " Quotas in /home and /tmp at the start:"
    quota
    news 2>/dev/null
    mail -p
  fi
#-------------------------------------------------------------------------
# --->    Best place for user-level commands
#
  ja $JOBACCT                       #start job accounting
#-------------------------------------------------------------------------
# --->    Execute actual job
```

```
#
  sh -ex >$STDOUT 2>$STDERR
#-------------------------------------------------------------------------
# --->    Job is done; job rundonw activities
#
echo >> $STDERR " Quotas in /home and /tmp at the end:"
quota >> $STDERR
sleep 10
if [ -f $STDERR ]
  then
  echo " "
  if [ x$ZIB_ORIGIN = xUSCP ]
    then
    cpsf  < $STDERR                #format stderr; shift one -->
    echo " "                       #force newline
    ja -cst $JOBACCT  | cpsf       #job accounting report
    jinfo             | cpsf       #job accounting report
    else
    cat $STDERR                    #format stderr
    echo " "                       #force newline
    ja -cst $JOBACCT               #job accounting report
    jinfo                          #job accounting report
    fi
  else
    echo " Unable to find output" ;
    echo " Don't execute rm -rf $TMPDIR in your script"
    echo " (since this is done by this profile anyway)"
  fi
  echo " "
  echo "1=============================================================================
  echo "                        stdout (standard output file)"
  echo " ============================================================================
  if [ x$ZIB_ORIGIN = xUSCP ]
    then
    cpsf < $STDOUT
    else
    cat $STDOUT
  fi
  echo " "
  if [ -f $STDASA ]
    then
    echo "1=============================================================================
    echo "                          stdasa (standard asa file)"
    echo " ============================================================================
    cat $STDASA
  fi
  echo " "                         #force newline
  echo " ============================================================================
  echo "                                  End of Job"
  echo " ============================================================================
  echo " "
  exit 1
else
  echo " "                         #force newline
#-------------------------------------------------------------------
```

```
# --->    Interactive only commands
#
    echo "Enter terminal type : \c"
  read reply
  TERM=$reply; export TERM
  quota
fi
```

3.6 Vorbesetzte Variablen

Jeder UNICOS-Job durchläuft vor dem Benutzer-Prolog (vgl. Kap. 3.5) den System-Prolog und NQS. Hier werden bereits eine Reihe von Variablen (vgl. Kap 7.3) vorbesetzt, die im Job verwendet werden können, aber nur dann verändert werden sollten, wenn man sich über die Konsequenzen im klaren ist.

Für den Benutzer mit der Benutzeridentifikation *btuserid* und dem Prozeß mit der Prozessidentifikation *780* werden z.B. folgende Variablen vordefiniert:

```
ALRJE=/tmp/alrje/btuserid                 # ALRJE-Katalog
ENVIRONMENT=BATCH                         # Batchjob
HOME=/home/btuserid                       # Heimatkatalog
JOBACCT=/tmp/780.btuserid/.JOBACCT        # Jobstatistik-Datei
LOGNAME=btuserid                          # Benutzeridentifikation
MAIL=/usr/mail/btuserid                   # Katalog für Mail
PATH=.:/sbin:/bin:/usr/bin:/usr/ucb:/usr/lbin:.  # Suchpfad für Kommandos
PERM=/bt/btuserid                         # permanenter Katalog (UNIX-Workstation serv01)
QSUB_HOST=sn118                           # Name des Rechners
QSUB_REQID=160.sn118                      # NQS Jobidentifikation
QSUB_REQNAME=first                        # NQS Jobname
QSUB_TZ=MEZ-1                             # Zeitzone
QSUB_WORKDIR=/usr/spool/uscp/jobs         # NQS Arbeitskatalog
SHELL=/bin/sh                             # Verweis auf Bourne Shell
SHELL_ID=780                              # Prozeßidentifikation
STDASA=/tmp/xxxxx/.STDASA                 # Standard-Datei für ASA-Ausgabe
STDERR=/tmp/xxxxx/.STDERR                 # Standard-Datei für stderr
STDOUT=/tmp/xxxxx/.STDOUT                 # Standard-Datei für stdout
TMPDIR=/tmp/xxxxx                         # temporärer Katalog
TZ=MEZ-1                                  # Zeitzone
ZIB_JSQ=10795                             # USCP-Jobidentifikation (falls der Job über eine
                                          # Station kam)
ZIB_MF=BE                                 # verwendete Station
ZIB_ORIGIN=USCP                           # Herkunft des Jobs: USCP über eine Station
ZIB_ORIGIN=ALRJE                          # Herkunft des Jobs: ALRJE
ZIB_TID=WR8A0                             # Terminal-Identifikation des zugeordneten Jobs auf
                                          # der Station
```

Die für den jeweiligen Job gesetzten Variablen erhält man durch Aufruf des Kommandos *env* ohne weitere Parameter.

3.7 Das UNICOS-Paßwort

Unter UNICOS wird - wie auch bei anderen Betriebssystemen üblich - der Zugang zu dem System durch ein Paßwort geschützt. Die zunehmende Vernetzung der Rechner mit der Möglichkeit, auch von entfernten, schwer identifizierbaren Orten auf einen Rechner zugreifen zu können, stellt an die Handhabung der Paßwörter durch Rechenzentrum und Benutzer zunehmende Sicherheitsanforderungen. Im Hinblick auf Auflagen, die das Rechenzentrum seinen Benutzern bei der Verwendung von Paßwörtern macht, muß eine Abwägung vorgenommen werden zwischen diesen Anforderungen und einer bequemen Arbeitsweise für den Benutzer. Dabei kann nicht immer ausgeschlossen werden, daß geforderte Maßnahmen vom Benutzer als lästig und für die konkrete Arbeit hinderlich empfunden werden. Die folgende Beschreibung soll dem Benutzer helfen, sich auf die Situation am ZIB besser einzustellen und damit Behinderungen seiner Arbeit nach Möglichkeit zu vermeiden.

Gültigkeitsdauer

Ein von Benutzer oder Betrieb eingerichtetes Paßwort ist an der UNICOS-Anlage des ZIB sieben Wochen gültig. Nach dieser Frist ist mit dem alten Paßwort kein regulärer Zugang zur Anlage mehr möglich, sie kann dann nur noch zu dem Zweck benutzt werden, das alte Paßwort durch ein neues zu ersetzen. Eine solche Änderung kann der Benutzer nicht nur erst bei Ungültigkeit eines Paßwortes durchführen, sondern bei Bedarf auch früher, jedoch nicht in den ersten zwei Wochen nach Änderung des Paßwortes. Innerhalb dieser Frist kann das Paßwort - falls erforderlich - nur vom Rechenzentrum geändert werden. Mit diesen Regelungen soll erreicht werden, daß ein unbemerkt bekannt gewordenes Paßwort nach nicht allzu langer Zeit in jedem Fall seine Gültigkeit verliert. Die Zwei-Wochen-Frist soll sicherstellen, daß ein Benutzer mit dem verständlichen Wunsch, immer das gleiche Paßwort zu haben, durch unmittelbares Rückändern des Paßwortes die angestrebte Wirkung nicht unterläuft.

Die Lebensdauer eines Paßwortes fällt in ein festes Wochenraster, welches jeweils auf einem Donnerstag liegt. Ist also ein Paßwort nur noch wenige Tage gültig, so ist im Laufe des nächsten Donnerstags mit dem Ablauf seiner Gültigkeit zu rechnen. Über die Anzahl der Tage, die das Paßwortes noch gültig ist, kann man sich durch Aufruf des Kommandos *pwval* informieren, welches folgende Meldung zeigt:

```
sn118$ pwval
--------------------------------------
Password is valid for  :       49 days.
Changing prohibited for:       14 days.
Last change of password: 19.04.90
No change before       : 03.05.90
--------------------------------------
Password will expire at: 07.06.90
--------------------------------------
```

Ein Aufruf von *pwval* befindet sich im ZIB-Standard-Profile, so daß der Benutzer damit aus jedem Standard-Ausgabefile die entsprechende Information entnehmen kann.

Das System prüft das Paßwort unmittelbar nach Eintreffen des Jobs auf der CRAY, und nicht erst dann, wenn dieser Job (bei hoher Betriebsmittel-Anforderung vielleicht erst einige Tage später) mit der Ausführung beginnt. Ablauf oder Änderung eines Paßwortes in der Zeit zwischen Entgegennahme und Ausführung eines Jobs ist damit für diesen Job bedeutungslos.

Regeln für die Neuvergabe des Paßwortes

Für vom Benutzer angegebene neue Paßwörter gelten folgende Regeln:

* Groß-/Kleinschreibung ist signifikant.

* Das Paßwort muß aus mindestens sechs Zeichen bestehen; nur die ersten acht Zeichen eines längeren Paßwortes werden ausgewertet.

* Es muß mindestens ein Sonderzeichen oder eine Ziffer enthalten.

* Es muß sich in mindestens drei Zeichen vom vorherigen alten Paßwort unterscheiden, wozu zwischen Groß- und Kleinschreibung nicht unterschieden wird.

Ändern des Paßwortes

Ein neues Paßwort kann vereinbart werden, indem die USER-Anweisung am Beginn eines Jobs um die Angabe des neuen Paßwortes erweitert wird:

```
# USER=userid PW=oldpw NPW=newpw
```

Diese Angabe führt zu einer Änderung des zuletzt gültigen Paßwortes *oldpw* in das neue Paßwort *newpw*. Sie darf frühestens zwei Wochen nach Einrichtung von *oldpw* gemacht werden und wirkt sowohl bei gültigem als auch bei abgelaufenem *oldpw*. Die Änderung des Paßwortes wird bereits bei Entgegennahme des Jobs durch die CRAY durchgeführt und nicht erst bei seiner Ausführung. Ist der Änderungsversuch nicht erfolgreich, z.B. weil eine der oben genannten Regeln verletzt wurde, erhält der Benutzer nach Entgegennahme des Jobs ein Ausgabe-File mit der Meldung:

```
uscpcmd: change of new passwd failed .
```

Der Job wird dann nicht ausgeführt und gelöscht. Im Falle der Fehlermeldung erfolgt die Zustellung des entsprechenden Ausgabe-Files vergleichsweise kurzfristig, so daß nach kurzer Zeit bei seinem Fehlen davon ausgegangen werden kann, daß die Paßwort-Änderung erfolgreich war.

Jeder Job, der nach Ablauf eines Paßwortes nicht eine erweiterte USER-Anweisung mit einem neuen Paßwort enthält, wird abgewiesen und gelöscht. Nach Entgegennahme des Jobs wird ein Ausgabe-File zugestellt mit der Fehlermeldung

```
uscpcmd: expired password.
```

Während bei abgelaufenem Paßwort nur die oben beschriebene Vorgehensweise möglich ist, kann bei noch gültigem Paßwort auch das Kommando *newpw* zur Änderung desselben verwendet werden. Der Aufruf erfolgt auf UNICOS-Kommando-Ebene mit dem Format

```
newpw 'oldpw' 'newpw'
```

wobei die Apostrophe zur Verhinderung von Veränderungen der eingegebenen Zeichenketten durch die Shell angegeben werden sollten.

Verhinderung der Wiedergabe von Paßwörtern

An Stellen im Job, an denen Paßwörter im Klartext vorkommen, wie es z.B. bei der Verwendung von *newpw* möglich ist, sollte eine Protokollierung lokal abgeschaltet werden, z.B. mit

```
set +xv
newpw 'oldloldl' 'new2new2'
set -x
```

Wo irgend möglich, sollte auf die Eintragung von Paßwörtern in Dateien ganz verzichtet werden.

Bei der Verwendung des Kommandos SUBCJ auf der NOS/VE-Anlage des ZIB gibt es eine gute Möglichkeit, welche die Eintragung des Paßwortes in den CRAY-Job überflüssig macht. Dabei wird die USER-Anweisung

```
# USER=userid PW=password
```

des Jobs, der z.B. den Filenamen *cjob* hat, abgewandelt in die Form

```
# USER=userid PW=@$PP('TYPE PASSWORD: ',true)@
```

Bei Abschicken das Jobs mit

```
SUBCJ cjob SM='@'
```

kann man nach Ausgabe der Aufforderung TYPE PASSWORD: das aktuelle Paßwort eingeben. Dabei wird das Echo automatisch abgeschaltet. Nach Drücken der Enter-Taste wird das Paßwort an der entsprechenden Stelle im Jobfile eingesetzt und der Job abgeschickt. Das Zeichen @ markiert den Bereich für die Ersetzung der Zeichenkette und kann variiert werden.

Änderung des Paßwortes im Dialog

Nach Ablauf eines Paßwortes erhält man beim Login-Versuch folgende Meldungen:

```
Your password has expired. Choose a new one
Old password:
New password:
Re-enter new password:
Connection closed by foreign host.
```

Die zweimalige Eingabe des neuen Paßwortes soll eine durch einen Tippfehler hervorgerufene falsche Eingabe verhindern.

3.8 Weitere Kommandos zur Arbeitsumgebung

3.8.1 Zugehörigkeiten anzeigen

logname	Das Kommando *logname* zeigt die *Login ID* an (entspricht $LOGNAME).
id	Das Kommando *id* zeigt die *UID* (User-Identification-Number) und die *GID* (Group-Identification-Number) des Benutzers sowohl als Nummern, wie sie dem System bekannt sind, als auch als mnemotechnischen Namen an.
groups [user]	Mit *groups* läßt sich feststellen, in welchen Gruppen man selbst (default) bzw. ein anderer Benutzer eingetragen ist.

3.8.2 Zeichenkette ausgeben

echo [arguments] Mit *echo* kann eine Zeichenkette nach *stdout* ausgegeben werden. Die Zeichenkette kann auch Steuerzeichen enthalten, z.B.:

\b Backspace
\f Form-feed
\n New-line
\t Tab

dabei auf die besondere Bedeutung von \ achten!

3.8.3 Nachrichten anschauen

Wichtige Mitteilungen des ZIB werden als einzelne Dateien im Katalog */usr/news* abgelegt. Mit dem Kommando *news* können deren Inhalte angesehen werden. Aktuelle Mitteilungen sind solche, deren Modifikationsdatum jünger ist als das der Datei *.news_time* im Heimat-Directory.

news [option] [items]

Optionen:

-a gib alle Nachrichten unabhängig von ihrem Alter aus (ohne Angaben von Optionen werden alle aktuellen, noch nicht gesehenen Nachrichten ausgegeben).

-n zeige eine Liste der aktuellen Nachrichten

-s zeige Anzahl der aktuellen Nachrichten an.

Die Nachricht *header* enthält aktuelle Informationen des ZIB. Im Standard-Prolog werden mit *news* alle für den jeweiligen Benutzer neuen Nachrichten ausgegeben.

3.8.4 Angaben zum Job-Accounting

Im ZIB-Standard-Prolog wird das Job Accounting durch *ja* initialisiert; es werden Informationen in der Accounting Datei *$JOBACCT* gesammelt. Vom Standard-Prolog her wird mit *ja -cs $JOBACCT* die Kommandostatistik und die Gesamtstatistik am Ende des Jobs ausgegeben. Zusätzlich können von dieser Datei durch zusätzlichen Aufruf von *ja* weitere Informationen ausgedruckt werden; die Ausgabe erfolgt auf *stdout*.

ja [options] [logfile]

wichtige Optionen zur Auswertung:

-c Kommando Statistik
-h Header
-s Gesamtstatistik
-l ausführliche Liste
-d Geräte spezifische I/O Statistik, falls vorhanden
-f Prozeßablauf
-t Terminieren des Job Accounting

Beispiel:

```
ja -l $JOBACCT
```

Hinweis: Das Kommando *ja* bewirkt ohne Parameter ein Einschalten (und damit ein Normieren) des Accountings, ist also bei Verwendung des Standardprofiles nur bedingt sinnvoll.

jinfo

Schreibt zusätzlich nützliche Informationen nach *stdout*, die von *ja* nicht geführt werden; im Standardprofile *.profile* bereits enthalten.

4. UNICOS-Kommandos

4.1 Kommandosyntax

In UNICOS wird, wie in anderen Betriebssystemen auf UNIX-Basis, jedes Kommando zunächst von einem Interpreter, der sogenannten *Shell*, gelesen und analysiert; diese Shell führt das Kommando dann selbst aus oder leitet es an ein geeignetes Programm weiter, das dann in einem eigenständigen Prozess abläuft. Die beiden bekanntesten Shells sind die *Bourne-Shell* und die *C-Shell* (oder auch Berkeley-Shell), die auch unter UNICOS verfügbar sind. Beim Arbeiten mit UNICOS im Batch über NQS (vgl. Kap. 3) ist die Bourne-Shell voreingestellt, die man - zumindest als UNIX-Anfänger - nicht verlassen sollte. Beide Shells haben einen umfangreichen Satz gemeinsamer Kommandos und in weiten Teilen die gleiche Syntax. So gelten auch die folgenden Ausführungen für beide Shells. Einzelheiten der Bourne-Shell sind im UNICOS User Commands Reference Manual, SR-2011, unter dem Stichwort sh(1) beschrieben.

4.1.1 Elementare Syntaxregeln

Ein einfaches Kommando besteht aus einem oder mehreren Wörtern, die durch Leerzeichen voneinander getrennt sind. Das erste Wort wird als Kommando- oder Programmname, die übrigen Wörter werden als Argumente aufgefaßt. Ein Argument kann eine Option, bestehend aus einem Schlüsselwort mit vorangestelltem "-"-Zeichen (ohne Leerzeichen) und eventuell einem zusätzlichen Parameter, oder ein Dateiname sein. Daraus ergibt sich für ein einfaches Kommando die folgende Form:

kommando -s1 p1 -s2 p2 ... file1 file2 ... .

Mehrere Schlüsselworte können hinter einem "-" zusammengefaßt werden, z.B.

kommando -s1s2s3 file.

Das Ende eines Kommandos wird durch den Beginn einer neuen Zeile oder durch ";" gekennzeichnet. Wird ein Kommando auf der nächsten Zeile fortgesetzt, muß das *linefeed (LF)* durch einen "\" (backslash) für die Shell außer Funktion gesetzt werden.

Beispiel:

```
kommando1; kommando2 -s1p1\
-s2p2 file.
```

Kommentare werden durch ein vorangestelltes "#" markiert und können an beliebiger Spaltenposition beginnen, z.B.

kommando1 file # Kommentar.

4.1.2 Kommandoketten (Pipes)

In UNICOS ist es, wie in allen UNIX-Systemen, möglich, die Ausgabe eines Kommandos direkt als Eingabe des nächsten Kommandos zu verwenden; der Weg über eine explizite Zwischendatei ist nicht notwendig. Dieses Verfahren nennt man *pipelining* und eine solche Kommandokette kurz *pipe*. Eine *pipe* hat z.B. folgende Form:

kommando1 | kommando2 | kommando3.

4.1.3 Umlenkung von Ein-/Ausgabe

In der Regel lesen Kommandos ihre Eingabedaten (falls vorhanden) von der Standardeingabe *stdin* und schreiben ihre Ausgabe auf die Standardausgabe *stdout* (Fehlermeldungen gehen entsprechend nach *stderr*, der Standardfehlerdatei). In einem Batchjob ist *stdin* nur über Dateien möglich; *stdout* und *stderr* führen das Ablaufprotokoll. Will man für diese Standardmedien explizit Dateien angeben, so ist das auf folgende Weisen möglich:

```
kommando < datei        # Kommando liest von datei,
kommando > datei        # Kommando schreibt auf datei, die bei Bedarf eingerichtet wird.
```

Beide Formen sind auch kombinierbar:

```
kommando < datei1 > datei2

kommando > > datei      # Kommando schreibt an das Ende von datei.

kommando < < endemarke  # Kommando liest alle folgenden Zeilen bis zu der, die
                        # ab der ersten Spalte endemarke enthält.
kommando 2> datei       # Kommando schreibt die Fehlermeldungen nach datei statt
                        # nach stderr.
```

4.1.4 Expansion von Parametern

Es gibt einige Zeichen, die von der Shell bei der Analyse von Kommandos auf besondere Weise interpretiert werden. Diese Metazeichen spielen besonders in Dateinamen eine wichtige Rolle:

*	Steht für eine beliebige Zeichenfolge (einschließlich den leeren String).
?	Bezeichnet genau ein beliebiges Zeichen.
[z1,z2,..]	Jedes der in den eckigen Klammern aufgeführten Zeichen wird erkannt. Es ist auch die Form [z1-z2] möglich, mit der ein Zeichen aus dem aufsteigend sortierten Bereich zwischen z1 und z2 beschrieben wird.
\	Der umgekehrte Schrägstrich (back slash) verhindert die Interpretation des folgenden Metazeichens durch die Shell.

Hinweis: Eine Reihe von Beispielen zur Interpretation von Metazeichen in Dateinamen sind im Kapitel 7 enthalten.

4.2 Übersetzung von Programmen

4.2.1 Der Aufruf eines FORTRAN-Übersetzers (CFT77, CFT)

Es stehen zwei FORTRAN-Compiler zur Verfügung: CFT77 und CFT (zum Vergleich der beiden Compiler und weiteren Übersetzungs-Möglichkeiten siehe Kapitel 8). Das FORTRAN-Quellprogramm wird aus einer Datei gelesen, deren Name mit *.f* enden sollte. Das übersetzte Binärprogramm wird auf die Datei *binfile* geschrieben, aus der mit Hilfe des Segmentladers SEGLDR ein lauffähiges Programm erzeugt werden kann. Die FORTRAN-Übersetzer CFT77 und CFT benutzen die spezielle Vektor-Hardware der CRAY X-MP, d.h. sie optimieren und vektorisieren innere DO-Schleifen. Die nachfolgend beschriebenen Aufrufe enthalten nur die wesentlichen Parameter; weitere Angaben sind in Kapitel 8 ausgeführt.

Der Aufruf des CFT77 lautet:

```
cft77 -b binfile -d offstr -e onstr -l listfile -o optim source.f
```

Optionen (die Voreinstellung (default) ist unterstrichen):

-b binfile source.o	Name der Datei, auf die die übersetzten Binärmodule geschrieben werden.
-d offstr	*offstr* ist eine Folge von Buchstaben. Jeder spezifizierte Buchstabe schaltet eine Übersetzer-Option aus.
-e onstr	*onstr* ist eine Folge von Buchstaben. Jeder spezifizierte Buchstabe schaltet eine Übersetzer-Option ein.
-l listfile source.l	Name der Datei, auf die die Programmliste geschrieben wird (Voraussetzung ist die Option *-e* mit einem der Werte *L, g, h, m, s* oder *x*).
-o optim full,zeroinc	Angabe der Übersetzer-Optionen für die Optimierung. Es können max. 2 Optionen in der Form *-o opt1,opt2* angegeben werden.
source.f	Name der Datei, die die FORTRAN-Quelle enthält.

Der Aufruf des CFT sieht formal genauso aus. Allerdings gibt es u. a. bei den Werten zu den Optionen *-d, -e* und *-o* einige Unterschiede (siehe Kapitel 8.3).

4.2.2 Der Aufruf des PASCAL-Übersetzers (PASCAL)

Der Übersetzer liest das PASCAL-Quellprogramm von der unter *-i* angegeben Datei *idn* oder von *stdin* (default). Das übersetzte Binärprogramm wird auf die Datei *bdn* geschrieben, von der es mit Hilfe des Segmentladers SEGLDR geladen werden kann. Der PASCAL-Übersetzer benutzt die spezielle Vektor-Hardware der CRAY X-MP, d.h. er optimiert und vektorisiert innere Schleifen und er bietet Spracherweiterungen zur Nutzung der Vektoroperationen.

Der Aufruf des PASCAL-Übersetzers lautet:

```
pascal -i idn -l ldn -b bdn -o list
```

Optionen (die Voreinstellung (default) ist unterstrichen):

-i idn stdin	Name der Datei, auf der sich die PASCAL-Quelle befindet.
-l ldn stdout	Name der Datei, die die Liste des Quellprogrammes enthält.
0	Unterdrückt eine Liste. "Fatal-"Fehlermeldungen werden auf *stdout* geschrieben.
-b bdn a.o	Name der Datei, auf die der Binärcode geschrieben wird.
-o list	Übersetzeroptionen können durch Kommata getrennt angegeben werden. Übersetzerdirektiven, die im PASCAL-Programm angegeben werden, überschreiben die Anfangsbesetzungen.

Anmerkung: PASCAL benötigt zur Übersetzung eines kleinen Programms 340000 Worte Hauptspeicher! Weitere Hinweise in Kapitel 9 und im PASCAL Reference Manual der Fa. CRAY (siehe Anhang B.1).

4.2.3 Der Aufruf der C-Übersetzer PCC und SCC.

Auf der CRAY X-MP des ZIB stehen zur Zeit zwei unterschiedliche C-Compiler zur Verfügung: Der bisher gewohnte C-Compiler auf Basis des Portable C-Compilers von AT&T sowie, seit April 1990, ein zusätzlicher C-Compiler, der dem vorgeschlagenen ANSI-Standard entspricht.

Der bisherige C-Compiler auf Basis von AT&T steht zur Zeit in der Version 4.1.7 bereit und wird vom Hersteller zwecks besserer Unterscheidung PCC (Portable C-Compiler) genannt.

Die für seine Anwendung maßgeblichen Handbücher sind:

SR-2024 CRAY C Reference Manual (Revision C),
SR-0136 C Library Reference Manual (Revision C).

Der zusätzlich eingeführte C-Compiler nach ANSI-Standard wird als Standard-C-Compiler bezeichnet (SCC), die Versionsnummer ist zur Zeit 1.0.4. Die für diesen Compiler maßgeblichen Handbücher sind:

SR-2074 CRAY Standard C Programmer's Reference Manual,
SR-2080 UNICOS Standard C Library Reference Manual.

Zu empfehlen ist weiterhin das Buch "The C Programming Language" in Englisch oder Deutsch von B. W. Kernighan und D. M. Ritchie.

Der **Aufruf des C-Compilers (PCC)** hat folgenden Aufbau:

```
cc options file.c
```

Dabei sind:

cc	Aufruf des C-Compilers mit Präprozessor und Lader,
options	Liste der zugehörigen Optionen.
file-name	Datei, die die C-Quelle enthält; die zwei letzten Zeichen des Namens müssen *.c* sein (Suffix).

Ohne Parameterangaben läuft das Kommando folgendermaßen ab:

Der Präprozessor (CPP) übernimmt die ganze C-Quelle aus der angegebenen Datei *file.c*, durchsucht sie nach CPP-Anweisungen (sie beginnen mit "#" in Spalte 1), führt sie aus und gibt seine Ausgabe als Eingabe an den C-Compiler. Der Compiler legt die fertige Übersetzung in eine Objektdatei *file.o*, die automatisch kreiert wird. Sie wird vom Lader SEGLDR übernommen, der die ausführbare Datei *a.out* erzeugt. Diese Datei enthält dann die übersetzte Quelle und den Inhalt der aufgerufenen Bibliotheks-Routinen. In den Ablauf von Präprozessor und Compiler kann durch einige Parameter eingegriffen werden (siehe SR-2024).

Diese kompakte Form der Bearbeitung von C-Programmen dient der vereinfachten Handhabung. Sollen die verschiedenen Schritte einzeln ablaufen, können über die Parameter und die Kommandos *cpp* und *segldr* sowohl Test als auch Ablauf allen Erfordernissen angepaßt werden, z.B. können mehrere Quelldateien, vorübersetzte Programmteile (in C, FORTRAN, Assembler) und verschiedene Bibliotheken verarbeitet werden.

Werden mehrere Quelldateien angegeben, entstehen auch mehrere Objektdateien. Dann kann das Laden nur mit dem SEGLDR und nicht im PCC-Lauf erfolgen. Der Programmablauf wird danach wieder mit dem Kommando *cc* eingeleitet. Die erforderliche Mindestspeichergröße beim PCC beträgt 425.000 Worte.

Der **Aufruf des Standard-C-Compilers (SCC)** erfolgt mit

scc options file.c

Die Beschreibung des Ablaufes entspricht im wesentlichen dem des Kommandos *cc* . Als Abweichung ist zu erwähnen, daß es keinen separat aufrufbaren Präprozessor gibt, daß das "#" für Präprozessor-Anweisungen nicht in Spalte 1 stehen muß und daß mehrere Quelldateien angegeben werden können (... *file1.c file2.c* ...). Übersetzter Code kann ebenfalls in eine Objektdatei abgelegt, ggf. mit anderen Modulen durch den SEGLDR verbunden und anschließend mit *scc* zum Ablauf gestartet werden.

Achtung: Groß-/Kleinschreibung ist relevant; die Anweisungen in C müssen in Kleinbuchstaben erfolgen. Als Sonderzeichen werden fast alle verfügbaren ASCII-Sonderzeichen verwendet. Weitere Hinweise in Kapitel 10.

4.3 Laden und Starten von Programmen durch SEGLDR

Der Segmentlader SEGLDR verarbeitet die vom Übersetzer erstellten Binärmodule, die von diesem in einer Datei abgelegt sein können oder sich in einer Bibliothek befinden. Die Module werden geladen und gebunden. Der Segmentlader SEGLDR ist entgegen seinem Namen ein vollwertiger und effizienter Lader sowohl für segmentierte als auch für nicht segmentierte Programme. Die Segmentierung von Programmen dient zur Verminderung des Hauptspeicherbedarfs: nicht der gesamte Programmcode und alle Datenbereiche, sondern nur Teile davon werden gleichzeitig im Hauptspeicher gehalten. Den Normalfall stellen nicht segmentierte Programme dar.

Neben dem *segldr*-Kommando steht unter UNICOS auch das Standard-UNIX-Kommando *ld* zur Verfügung. *ld* ist kein eigenständiges Ladeprogramm, sondern ruft den SEGLDR auf. Es bietet dafür das in UNIX übliche Kommandozeilenformat und die üblichen Voreinstellungen (genaueres mit dem Kommando *man ld*). Bei Verwendung des Laders *ld* können SEGLDR-Direktiven (s.u.) mit der Option

-D dirstring

an den SEGLDR weitergereicht werden.

Der Lader ld wird seinerseits von den Übersetzungssystemen .i.cc;, .i.cf ;und .i.cf77 ;aufgerufen. Bei Verwendung dieser Kommandos können Ladedirektiven mit der Option

-d dirstring

an den SEGLDR durchgereicht werden.

Der Aufruf des SEGLDR (mit einer Auswahl von Parametern) lautet:

segldr -f value -g -i dirfiles -l names -m -o outfile -t -D dirstring -L ldirs -M file,opts -N -V objfiles

Optionen (die Voreinstellung (default) ist unterstrichen):

-f value	Parameter zur Steuerung der Speichervorbesetzung.
indef	Der Speicher wird mit einem Bitmuster vorbesetzt, das Gleitkommaüberlauf anzeigt, so daß beim Rechnen mit nicht vorbesetzten Gleitkomma-Variablen ein Jobabbruch wegen Gleitkommafehler erfolgt.
zeros	Speichervorbesetzung mit binären Nullen.
ones	Speichervorbesetzung mit -1 (alle Bits auf 1 gesetzt).

-g	Erzeugt eine DEBUG-Symboltabelle (kann entfallen, wenn die beteiligten Module debugfähig übersetzt wurden).
-i dirfiles	Namen der Dateien, von denen Direktiven gelesen werden.
-l names	Namen der Bibliotheken, die zusätzlich zu den Systembibliotheken beim Laden berücksichtigt werden. Beginnt der Name mit "." oder "/", so muß der volle Pfadname angegeben werden; andernfalls wird bei der Angabe jedes Bibliotheksnamens der Ausdruck */lib/lib* bzw. */usr/lib/lib* davorgesetzt und der Ausdruck *.a* dahinter, so daß mithin zuerst z.B. nach */lib/libname1.a* gesucht wird und dann nach */usr/lib/libname1.a* . Der erste zum gesuchten Eingangsnamen passende Modul wird genommen, d.h. die Reihenfolge der angegebenen Bibliotheken ist relevant. Mehrere Bibliotheksnamen sind durch Kommata zu trennen.
-m	Die Ladeliste wird nach *stdout* geschrieben (identisch mit *-M,address* , s.u.).
-o outfile a.out	Name der Datei, auf die das absolute (binäre) Objektmodul geschrieben wird. Fehlt dieser Parameter und auch die ABS-Direktive, so wird die Datei *a.out* erzeugt.
-t	Es wird ein Testlauf durchgeführt (vgl. Direktive TRIAL).
-D dirstring	Anweisungen für SEGLDR können als Direktiven *dirstring* bereitgestellt werden. Die Direktiven werden als erste Anweisungen der Eingabe-Datei ausgeführt. Die einzelnen Anweisungen werden durch ein Semikolon getrennt (Beschreibung einer Auswahl von Anweisungen siehe unten).
-L ldirs	Namen der Kataloge, in denen die unter *-l* angegebenen Bibliotheken gesucht werden sollen (ändert also die Voreinstellung */lib* und */usr/lib*).
-M file, opts stdout,a	Name der Datei für die Druckausgabe; fehlt *file*, wird nach *stdout* ausgegeben. Die Optionen *opts* steuern den Umfang der Ausgabe; die Angaben entsprechen denen der Direktive MAP (in Kleinbuchstaben!); fehlen *opts*, wird *address* ausgeführt und es ist die Angabe von *file* notwendig.
-N	Es wird keine Systembibliothek durchsucht.
-V	Die Identifikationszeile des SEGLDR wird nach *stderr* geschrieben.
objfiles	Namen der Dateien, die die Objektmodule enthalten. Konvention: Suffix *.o* haben Dateien, die von Compilern erzeugt wurden, Suffix *.a* solche, die von BLD erzeugt wurden (Modulbibliotheken).

Beispiel a.):

```
segldr -i segdir test.o       # Laden, Binden
a.out                         # Ausführen
```

Beispiel b.):

```
segldr -m -D'LIB=lib1,lib2;PRESET=ZEROS;MLEVEL=NOTE'\
test.o                        # zum Direktivenstring s.u.
a.out << ZZ                   # Ausführen
.
(Daten)
.
ZZ
```

4.3.1 Die Anweisungen für den Segmentlader

Die Anweisungen für den Segmentlader sind nachfolgend aufgelistet, soweit sie nicht der Beschreibung der Segmentstruktur dienen; die wichtigsten Anweisungen sind näher erläutert. Kommentare werden durch * eingeleitet. Anweisungen werden durch Semikolon, Zeilenende oder * beendet, Listenelemente durch Kommata getrennt. Anweisungen, die Listen enthalten, können nach einem Komma auf der nächsten Zeile fortgesetzt werden. Alle Anweisungen können in Groß- oder Kleinbuchstaben (aber nicht gemischt) geschrieben werden. Weitere Angaben und Einzelheiten sind dem SEGMENT LOADER (SEGLDR) Reference Manual SR-0066 zu entnehmen (siehe Anhang B.1).

4.3.2 Dateizuordnung

Mit den Anweisungen BIN, LIB, NODEFLIB und ABS werden Dateien definiert, die der SEGLDR erzeugen und/oder benutzen soll.

BIN = dn1,dn2, ...,dnn	Name der binären Eingabe-Dateien. Bei Namensgleichheit wird das erste Modul geladen; alle weiteren Modulnamen werden in einer Meldung ausgedruckt.
LIB = lib1,lib2, ...,libn	Namen der Bibliotheken, die zusätzlich zu den Systembibliotheken beim Laden berücksichtigt werden sollen. Der Segmentlader sucht und lädt nur die benötigten Module einer Bibliothek. Bei der Auflösung von Referenzen werden zuerst die bei BIN angegebenen Dateien, dann die bei LIB angegebenen Bibliotheken in der spezifizierten Reihenfolge und erst dann die Systembibliotheken durchsucht. Treten aufgrund der angegebenen Bibliotheken Namenskonflikte auf, müssen evtl. die Systembibliotheken bei LIB mit angegeben werden, um das Programm aus der gewünschten Bibliothek zu laden. Bei Namensgleichheit wird das zuerst gefundene Modul geladen; über weitere gefundene Modulnamen werden keine Meldungen ausgegeben.
NODEFLIB	Ignoriert alle voreingestellten Bibliotheken.
ABS = dn = a.out	Datei, auf die der absolute Objektmodul geschrieben wird.

4.3.3 Listensteuerung

Mit den Anweisungen ECHO, MAP und TRIAL wird die Ausgabe von SEGLDR auf die Ausgabedatei gesteuert.

TRIAL	führt einen Testlauf durch, ohne daß eine Ausgabe erzeugt wird. Man erhält dadurch die Ladeliste, die meisten Fehlermeldungen sowie Angaben zum Speicherbedarf.
ECHO = OFF = ON	unterdrückt die Druckausgabe von Eingabe-Anweisungen bzw. setzt den Ausdruck fort.
MAP = dir	Durch die Zuweisung unterschiedlicher Parameter zur MAP- Anweisung kann man sich unterschiedliche Daten aus der Ladeliste des SEGLDR ausgeben lassen.
= NONE	Es werden keine Ladelisten ausgegeben.
= STAT	Angaben zum gebundenen Programm wie Größe, Datum und Uhrzeit.
= ALPHA	Wie STAT, zusätzlich alphabetisch sortierte Ladeliste für alle Unterprogramme.
= ADDRESS	Wie ALPHA, nach Adressen sortiert.
= PART	Wie ALPHA plus ADDRESS.

=EPXRF	Entry Point Crossreference: wie STAT, zuzüglich Kreuzreferenzliste der Eingangspunkte.
=CBXRF	Common Block Cross Reference: wie STAT, zuzüglich Kreuzreferenzliste der COMMON-Blöcke.
=FULL	Wie PART plus EPXRF plus CBXRF.
=BRIEF	Wie ADDRESS plus ALPHA, jedoch begrenzt auf Module aus binären Eingabe-Dateien.

4.3.4 Module und gemeinsame Speicher-Blöcke (Common-Blocks)

Mit den Anweisungen MODULES, COMMONS und DYNAMIC kann die Reihenfolge festgelegt werden, in der die zu ladenden Module oder (gemeinsame) Speicherblöcke geladen werden.

MODULES=mod	Einzelne Module können angegeben werden, die zu binden sind. Mehrere Angaben sind durch Kommata zu trennen.
:dn	
=mod	Das Modul *mod* soll aus der ersten Datei, in der er gefunden wird, gebunden werden.
=mod:dn	Das Modul *mod* soll aus der Datei *dn* gebunden werden, auch wenn er in mehreren Dateien vorkommt.

COMMONS=blk	Es werden COMMON-Blöcke in der angegebenen Reihenfolge gebunden. Mehrere Angaben sind durch Kommata zu trennen.
:size	
=blk	Benennt den zu bindenden COMMON-Block.
=blk:size	Überschreibt zusätzlich die in den Programmquellen angegebene Größe des COMMON-Blockes.

DYNAMIC=blk	Legt den Anfang des (gemeinsamen) Speicherblocks auf das erste Wort, das dem längsten Segment-Abschnitt folgt. Die Größe dieses (gemeinsamen) Speicherblocks kann vom Benutzer verändert werden.
=//	Definiert den unbenannten (gemeinsamen) Speicherblock ("Blank Common") als dynamisch.

4.3.5 Fehlerbehandlung

Mit den Anweisungen MLEVEL und USX wird die Fehlerbehandlung gesteuert.

MLEVEL=ERROR	Nur Fehlermeldungen werden ausgegeben; SEGLDR beendet sich augenblicklich. Es wird kein ausführbares Programm erstellt.
=WARNING	Fehler- und Warnungsmeldungen werden ausgegeben. Zwar wird kein ausführbares Programm erstellt, die Verarbeitung wird aber fortgesetzt, so daß weitere Meldungen ausgegeben werden können.
=CAUTION	Ausgabe von Fehler-, Warnungs-, und Achtungmeldungen. Ein ausführbares Programm wird trotz aufgetretenen Fehlers erzeugt.
=NOTE	Ausgabe von Fehler-, Warnungs-, Achtung- und Hinweismeldungen. SEGLDR wurde z.B. nicht korrekt oder ineffektiv benutzt; keine Auswirkung auf die Ausführung.
=COMMENT	Alle Meldungen werden ausgegeben. Keine Auswirkung auf die Ausführung.

USX=WARNING	SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt kein ausführbares Programm.
=CAUTION	SEGLDR kennzeichnet unbefriedigte Externbezüge (unsatisfied external symbols) und erzeugt ein ausführbares Programm.
=IGNORE	SEGLDR erzeugt trotz unbefriedigter Externbezüge (unsatisfied external symbols) ohne Warnung ein Programm.

4.3.6 Speichersteuerung

Mit den Anweisungen HEAP und STACK kann die Größe der Speicherbereiche festgelegt werden, die durch die vom System angebotene Heap- und Stackverwaltung dynamisch zur Laufzeit verändert werden kann. Falls die DYNAMIC-Anweisung angegeben ist, ist die Größe des Heap nicht dynamisch veränderbar.

HEAP = init	Anfangsgröße in Worten für die Heap-Verwaltung.
= init + inc	Increment: Größe der Speichererweiterung, wenn der Heap überläuft. Durch eine DYNAMIC-Anweisung (s.o.) wird *inc* auf 0 gesetzt.
= init > min	Minimum: gibt die kleinste Blockgröße (*min* größer oder gleich 2) in der Freispeicherliste des Heap an.
= init + inc > min	Kombination aus *inc* und *min*.
= 2048 + 2 > = 2	Wird eingesetzt, wenn die Heap-Anweisung ohne Parameter angegeben ist.
STACK = init	Anfangsgröße in Worten für den Teil des Heap, der der Stack-Verwaltung zur Verfügung gestellt wird.
= init + inc	Increment: Größe der Speichererweiterung, wenn der Stack überläuft.
= 2048 + 256	Wird eingesetzt, wenn die Stack-Anweisung ohne Parameter angegeben ist.

4.3.7 Vorbelegen von Speicherbereichen

Mit der Anweisung PRESET lassen sich Speicherbereiche vorbelegen:

PRESET = ONES	Vorbelegung mit -1 (Achtung: Plural!!).
= ZEROS	Vorbelegung mit 0 (Achtung: Plural!!).
= INDEF	Vorbelegung mit einem Wert derart, daß dessen Verwendung in einer Gleitkomma-Operation zur Anzeige eines Gleitkomma-Fehlers führt (Achtung: Abweichung vom CRAY Standard!).
= -INDEF	Vorbelegung wie *indef*, aber negativ.
= value	Eine 16-bit Oktalzahl kann vom Benutzer vorgegeben werden.

4.3.8 Ersetzen von entry points

EQUIV = epname(syn_1,syn_2, ... ,syn_n) Anweisung an den SEGLDR, den Namen von *entry points* (syn_n) durch einen anderen Namen (*epname*) zu ersetzen. Die Wirkung entspricht dem globalen Ersetzen im Programm per Editor.

Beispiel:

```
      .
      .
      .
      CALL A
      .
      .
      .
      CALL B
      .
      .
      .
```

Durch *EQUIV = C(A,B)* werden vom SEGLDR die Aufrufe sowohl von *A* als auch von *B* durch jeweils einen Aufruf von *C* substituiert.

4.4 Bearbeiten von Objektbibliotheken (BLD)

Mit BLD können Objektbibliotheken erstellt, verändert und erweitert werden. Der Unterschied zwischen sequentiell auf einer Datei stehenden Modulen und einer Bibliothek ist der, daß am Ende einer Bibliothek ein Inhaltsverzeichnis steht. Die Bedeutung der Bibliotheken liegt nun darin, daß der Lader im Stande ist, nur die Module herauszusuchen, die benötigt werden. Bei einer sequentiellen Datei werden alle Module geladen. Der Lader durchsucht dabei das Verzeichnis der Bibliothek und stellt so sehr schnell fest, ob ein Modul vorhanden ist und an welcher Stelle es in der Datei steht. Das Verwaltungsprogramm wird durch das Kommando *bld* aufgerufen. Eine Bibliothek kann aufgebaut, verändert oder aufgelistet werden. In den meisten Fällen werden nur wenige Parameter benötigt.

Der Aufruf des BLD hat folgende Form:

```
bld key[opts] position_obj bldname args
```

Ein zweites Format wird ausschließlich beim Umbenennen von Modulen verwendet und ist am Ende dieses Kapitels als Beispiel angegeben. Durch das Kommando *man bld* erhält man die vollständige Beschreibung, die auch einige Beispiele umfaßt.

Die Parameter haben folgende Bedeutung:

key	Buchstabe, der die Funktion im wesentlichen beschreibt.
opts	Ein oder mehrere Buchstaben, die als Option die durch *key* gewählte Funktion ergänzen; *key* und *opts* bilden eine Zeichenfolge (ohne Blank!).
position_obj	Name eines Moduls; die Angabe ist nur notwendig, wenn durch *opts* eine Position innerhalb der Bibliothek festgelegt wird.
bldname	Name der Bibliothek.
args	Namen von Dateien, die die zu bearbeitenden Moduln enthalten, oder Modulnamen; der Typ dieses Parameters hängt von *key* ab.

Auswahl einiger der bei *key* und *opts* möglichen Parameterwerte:

key:	
d	Löscht die unter *args* angegebenen Module in der Bibliothek *bldname*.
q	Erzeugt aus den unter *args* angegebenen Dateien eine neue Bibliothek *bldname*; es werden alle Module übernommen (gleichnamige kommen mehrfach vor).
r	Die Module aus den unter *args* angegebenen Dateien werden in die Bibliothek *bldname* übernommen; dabei werden gleichnamige Module überschrieben; falls die Datei *bldname* noch nicht existiert, wird sie neu eingerichtet.
t	Ein Inhaltsverzeichnis der Bibliothek *bldname* wird ausgegeben.
x	Die unter *args* angegebenen Module werden in gleichnamige Dateien ausgegeben.
opts:	
a, b, i	Die durch *args* spezifizierten Module werden hinter (*a*) bzw. vor (*b* oder *i*) dem Modul *position_obj* in die Bibliothek eingefügt; Voraussetzung ist die Angabe von *r* als *key*.
z	Durch diese zusätzliche Option wird erreicht, daß die Module in der Bibliothek nur mit den zugehörigen Programmnamen bezeichnet werden und nicht zusätzlich als Präfix den Namen der Datei erhalten, aus der sie stammen. Bei Moduln, die aus Dateien in $TMPDIR stammen, ist dieses Präfix bei späteren Modifikationen der Bibliothek hinderlich.

Anlegen und Verwenden einer Binär-Bibliothek

Im folgenden Beispiel wird eine neue Bibliothek angelegt. CFT77 schreibt die Module auf die Datei *quelle.o*. Diese Datei wird als Eingabe-Datei genommen und auf die Ausgabe-Datei, die neue Bibliothek *zuglib*, kopiert. Die Bibliothek ist nun angelegt und kann vom Lader verwendet werden.

Beispiel (Bibliothek anlegen):

```
# USER=(userid) PW=(password)
# QSUB-r bldgen
# QSUB-lm 1Mw
# QSUB-lT 6
# QSUB-ZIB S1
cd $TMPDIR              # Wechsel zum $TMPDIR directory
cat > quelle.f << '%'
SUBROUTINE LOK
...
SUBROUTINE PACKW
...
%
cft77 -eas quelle.f     # cft77 erzeugt quelle.o
cd                      # Rückkehr nach $HOME
bld qz zuglib $TMPDIR/quelle.o
cp zuglib $PERM         # sichern der zuglib auf PERM-Bereich
exit
```

Für den Zugriff auf die im obigen Beispiel erzeugte Bibliothek muß im Kommando *segldr* deren Name als Direktive angegeben werden; diese Bibliothek wird dann vom SEGLDR vor den Systembibliotheken als erste durchsucht (z.B. nach dem Mudul LOK).

Beispiel (Bibliothek verwenden; ohne Jobvorspann):

```
...
cd $TMPDIR
cat > quelle.f << %
PROGRAM BAHN
...
CALL LOK
...
END
%
cft77 -eas quelle.f
segldr -D "lib=$PERM/zuglib" quelle.o
a.out
exit
```

In dem FORTRAN-Programm wird das Unterprogramm LOK aufgerufen. Der Lader lädt nur das Modul LOK aus der Bibliothek.

Verändern einer Binär-Bibliothek

In der vorhandenen Bibliothek *zuglib* soll ein Modul durch ein gleichnamiges, neu erzeugtes Modul ersetzt werden (*replace*). Bis auf den Schlüssel *r* entspricht das Beispiel dem oben angegebenen zum Anlegen einer Bibliothek.

Beispiel:

```
...
cd $TMPDIR
cat > quelle.f << %
SUBROUTINE LOK
...
SUBROUTINE KLASSE1
...
%
cft77 -eas quelle.f
cd $PERM
bld rz zuglib $TMPDIR/quelle.o
```

Erstellen einer Bibliothek aus vorhandener Bibliothek und neu generierten Moduldateien (file.o):

Um aus Modulen in einer vorhandenen Bibliothek und neuen, durch Übersetzerläufe erzeugten Moduldateien eine neue Bibliothek aufzubauen, gibt es grundsätzlich zwei Möglichkeiten: Man kann

1.) in einer Kopie der vorhandenen Bibliothek die nicht benötigten Module löschen und die neuen ergänzen oder

2.) die zu erhaltenden alten Module zunächst in einzelnen Dateien (mit dem Namen der Module) ablegen und aus diesen dann zusammen mit den neuen Moduldateien die neue Bibliothek aufbauen.

Das folgende Beispiel beschreibt den zweiten Weg:

Beispiel:

```
...                              #working directory sei $HOME
mkdir libgen
cd libgen
cft77 -eas $PERM/quelle.f        #neue Module generieren
bld x $PERM/libalt               #alle Module aus libalt werden als
                                 #Dateien unter dem jeweiligen Modulnamen
                                 #abgelegt, und zwar mod1, mod2, mod3
bld q $PERM/libneu quelle.o mod1 mod3
                                 #von quelle.o werden alle Module, ferner
                                 #mod1 und mod3 übernommen
```

Umbenennen von Modulen

Beim Umbenennen von Modulen sind die Regeln für die Namensgebung zu beachten; die allgemeine Form ist

dateiname:modulname.

Im folgenden Beispiel wird angenommen, daß der Name des Moduls in der Bibliothek ohne den Namen der ihn enthaltenden Datei erzeugt wurde (der führende Doppelpunkt ist signifikant!):

Beispiel:

```
bld R zuglib :lok :lok1        # Modul lok wird auf lok1 umbenannt
```

4.5 CRAY-spezifische Unterprogramme

In diesem Abschnitt werden einige systemnahe Unterprogramme aufgeführt, die in speziellen Systembibliotheken liegen und die nicht zum FORTRAN Standard gehören. Im Kapitel 12.7 sind die wichtigsten Routinen der Bibliothek für wissenschaftliche Anwendungen (SCILIB) aufgeführt. Eine vollständige und ausführliche Dokumentation aller Routinen findet man im UNICOS Math and Scientific Library Reference Manual (SR-2081).

Diese Routinen stehen ohne besondere Maßnahmen jedem Benutzerprogramm zur Verfügung, da sie in Systembibliotheken residieren, die beim Laden (SEGLDR) automatisch benutzt werden. Benutzerprogramme mit gleichen Namen haben beim Laden Vorrang, da der Lader die Systembibliotheken standardmäßig als letzte durchsucht. Werden die Systemroutinen gewünscht, obwohl in anderen beim Laden angegebenen Bibliotheken Routinen gleichen Namens vorhanden sind, so ist durch eine geeignete Reihenfolge sicherzustellen, daß die Systemroutinen verwendet werden.

System-Hilfsroutinen

CLEARFI — CALL CLEARFI unterdrückt Gleitkomma-Unterbrechungen und verhindert so die Feststellung von *Zerodivide* und *Overflow*. CALL CLEARFI macht unter Umständen die Vektorverarbeitung einer darauf folgenden Schleife erst möglich.

SETFI — CALL SETFI schaltet o.g. Zustand wieder aus.

Beispiel:

```
      CALL CLEARFI
      DO 1 I=1,N
      Y(I)=1.
      IF (X(I) .NE. 0.) Y(I)=SIN(X(I))/X(I)
1     CONTINUE
      CALL SETFI
      .....
cft77 -es quelle.f
```

SMACH — Abfragen der Maschinenkonstanten und der kleinsten oder größten darstellbaren Zahl.
Aufruf: SMACH(int);

Int = 1. Resultat: 7.105E-15 (kleinste Zahl "Epsilon", so daß 1.0 ± "Epsilon" ungleich 1 ist)
Int = 2. Resultat: 1.290E-2450 (kleinste Zahl größer als 0)
Int = 3. Resultat: 7.750E + 2449 (größte Zahl)

TRBK — Traceback als Testhilfe. CALL TRBK bewirkt eine Ausgabe nach *stderr*, aus der unter anderem die Aufrufstelle (Unterprogramm, Block) hervorgeht.

DATE — Aktuelles Datum. Aufruf: CALL DATE(*date*). Man erhält das aktuelle Datum in der amerikanischen Reihenfolge *mm/dd/yy*.

GDATE	*German Date:* aktuelles Datum. Aufruf: CALL GDATE(*date*). Man erhält das aktuelle Datum in der deutschen Reihenfolge *dd.mm.yy*.
JDATE	Aktuelles Julianisches Datum. Aufruf: CALL JDATE(*date*). Man erhält das Julianische Datum in der Form *yyddd*.
CLOCK	Aktuelle Uhrzeit. Aufruf: CALL CLOCK(time). Man erhält die Uhrzeit in der Form *hh:mm:ss*.
TIMEF	Realzeitdifferenz. Aufruf: CALL TIMEF(R). Man erhält die Realzeitdifferenz zwischen dem aktuellen Aufruf und dem ersten Aufruf von TIMEF in Millisekunden.
SECOND	Verbrauchte CPU-Zeit. Aufruf: CALL SECOND(R). Man erhält die bisher verbrauchte CPU-Zeit des Jobs in Sekunden.
TREMAIN	Verbleibende CPU-Zeit. Aufruf: CALL TREMAIN(R). Man erhält die Restzeit bis zum Ablauf der Jobzeit in Sekunden.

4.6 Aufruf von UNICOS-Kommandos aus FORTRAN-Programmen

Jedes UNICOS-Kommando kann von FORTRAN-Programmen aus aufgerufen werden. Im *Function*-Unterprogramm **ISHELL** kann eine Kommandofolge, in Apostrophe eingeschlossen, als Parameter angegeben werden.

Hinweis: für den Aufruf der Shell muß der vom Programm aktuell belegbare Speicherplatz zunächst verdoppelt werden ("fork"), was u.U. die zugewiesenen Resourcen übersteigt; d.h., bereits nur ein *ls* kann unangenehme Nebeneffekte haben!

Beispiel:

```
ISTAT = ISHELL('cd $HOME; ls -alF')
```

ISTAT erhält im Fehlerfall einen Wert ungleich 0. Das Unterprogramm ist im UNICOS Math and Scientific Library Reference Manual (SR-2081) beschrieben.

SGSHELL - eine Alternative zu ISHELL

Die vorübergehende Verdopplung des Speicherplatzes ("fork", s.o.) bei ISHELL kann durch die Verwendung von **SGSHELL** vermieden werden. Das auszuführende Kommando wird dazu in eine sog. *named pipe* geschrieben, die von einer im Hintergrund laufenden Shell gelesen wird. Das vorhandene FORTRAN-Programm muß für den Aufruf von SGSHELL nicht geändert werden. Den Einsatz von SGSHELL in einem Batchjob zeigt das folgende einfache Beispiel:

```
# USER=userid PW=password
# QSUB-r sgshell
# QSUB-lT 6
# QSUB-lm 1Mw
# Test mit SGSHELL aus FORTRAN-Programm
#
cd $TMPDIR
cat > progf.f << %
      program cmd
```

```
      integer err
c
      err=ishell('cd $HOME; ls -al')
c
      if (err .lt. 0) then
         write(6,100) err
 100     format(' +++ fehler in cmd: ',i3)
      else
         write(6,200) err
 200     format(' *** cmd ls executed, errcode: ',i3)
      endif
      stop
      end
%
cft77 progf.f

segldr -D"EQUIV=SGSHELL(ISHELL)" progf.o

ishd.sh   # Script für Start einer "server shell", das
          # unter "ishd.sh" allgemein verfügbar ist
a.out
```

Hinweis:

Bei mehrfachen Programmläufen muß das Script *ishd.sh* jedesmal vorher aufgerufen werden; fehlt der Aufruf, so erhält man den Exitstatus -1.

4.7 Analysieren von Dateien (cmp, diff, od, nm)

Vergleich von Dateien (*cmp* und *diff*)

Vergleich zweier binärer Dateien

```
cmp -l -s file1 file2
```

Parameterauswahl:

-l	Ausgabe der Byte-Nummer (dezimal) und der beiden unterschiedlichen Bytes (oktal) in drei Spalten nach *stdout*; Default: keine Ausgabe.
-s	Ausschließliche Ausgabe des *Exit Codes* bei unterschiedlichen Dateien; Default: keine Ausgabe.

Beispiel:

```
cmp -l file1 file2
```

(Zum *exit status* vergleiche Hinweis zu *diff*).

Vergleich zweier Textdateien

```
diff -efbh file1 file2
```

Das Kommando *diff* vergleicht zwei Textdateien miteinander und erzeugt (Parameter *-e*) Änderungsanweisungen für den Editor *sed* (bzw. *ed*). Die Anweisungen sind nach Zeilennummern absteigend sortiert und können folgende Form haben:

n a text .	Hinter Zeile *n* wird *text* eingefügt; der "." in der ersten Spalte ist das Endekriterium.
n d	Zeile *n* wird gelöscht.
n c text .	Zeile *n* wird gelöscht und nachfolgend *text* eingefügt; der "." in der ersten Spalte ist das Endekriterium.

Beispiel:

```
diff -eb text1 text2 > ed.dir || cat ed.dir
```

Hinweis: Das Kommando *diff* endet mit

exit status 0	falls keine Unterschiede,
exit status 1	falls Unterschiede,
exit status 2	falls sonstige Fehler festgestellt wurden; d.h. die Datei *ed.dir* enthält entweder die Änderungsanweisungen oder eine interne Fehlermeldung.

Datei-Dump (*od*)

Mit dem Kommando *od* kann eine Datei ganz oder teilweise symbolisch aufgelistet werden. Der Inhalt der Datei wird Wort für Wort oktal, hexadezimal, als Integerzahl mit oder ohne Vorzeichen sowie als ASCII-Zeichenkette interpretiert. Durch eine byte- oder blockorientierte Adresse kann (oktal oder dezimal) die Stelle angegeben werden, an der die Ausgabe beginnen soll.

od -bcdosxpBCW file offset b

Parameter (Auswahl):

-o	(Default:) Ausgabe wortweise oktal.
-c	Ausgabe byteweise als ASCII-Zeichenkette, Sonderzeichen in der Darstellung gemäß der Programmiersprache C.
-x	Ausgabe wortweise hexadezimal.
-C	Bei Parameter *-d, -o, -s* oder *-x* zusätzliche Ausgabe einer Spalte, in der die Bytes als ASCII-Zeichen interpretiert werden.

Beispiele:

```
od -c test.f            #Ausgabe der ganzen Datei als
                        #ASCII-Zeichenkette
od -xC test.f 200       #Ausgabe ab Byte 200 oktal
od -c test.f 136.       #Ausgabe ab Byte 136 dezimal
```

Struktur einer Objektdatei (*nm*)

Das Kommando *nm* gibt für eine oder mehrere Objektdateien eine Liste der externen Namen und der Eingangsnamen (Symboltabelle) aus. Die externen Namen werden durch *U* (undefined) und die Eingangsnamen durch *T* (text segment symbol) gekennzeichnet.

```
nm -gaoux file
```

Parameter (Auswahl):

-g	Ausgabe von externen Namen und Eingangsnamen (Parameter notwendig!).
-o	Zusätzliche Ausgabe des Modulnamens am Anfang jeder Zeile.
-u	Ausschließliche Ausgabe von externen Namen (Eingangsnamen werden unterdrückt).

Beispiel:

```
nm -g test.o
```

4.8 Beenden von Jobs von der CRAY aus

Mit dem Kommando *deljob* kann der Benutzer von der CRAY aus seine eigenen Jobs löschen. Als Identifikation für den zu löschenden Job kann der *jobname* oder die *nqsnr*, die mit RST auch auf den Vorrechnern (siehe Kapitel 5.6) ausgewiesen wird, angegeben werden. Es können nur Jobs gelöscht werden, die der gleichen Benutzerkennung angehören wie der des aufrufenden Jobs.

Sollten sich mehrere Jobs des Benutzers mit dem gleichen Jobnamen in der CRAY befinden, sollte nur unter Angabe der *nqsnr* der Auftrag zum Löschen gegeben werden, da sonst eventuell der falsche Job gelöscht wird.

Aufruf:

deljob -jn jobname -nq nqsnr

jobname	Der *jobname* entspricht der Angabe, die im *# QSUB -r* -Kommando des Jobs angegeben worden ist. Benutzer, die den zu löschenden Job per ALRJE zur CRAY geschickt haben und keine Angabe zu *# QSUB -r* gemacht haben, müssen den Dateinamen der Datei angeben, in der der Job gestanden hat, da dieser als *jobname* genommen worden ist.
nqsnr	Die fünfstellige *nqsnr*, die im RST ausgewiesen wird; diese ist eindeutig.

Zur Identifizierung des Jobs genügt *eine* Angabe - entweder *jobname* oder *nqsnr*.

4.9 Online-Dokumentation

Wie in UNIX-Systemen üblich, sind auch in UNICOS wesentliche Teile der Beschreibung über das Kommando *man* online verfügbar (sog. *man-pages*). Das Kommando schreibt nach *stdout*, durch Ausgabeumlenkung also auch in eine beliebige Datei. In den folgenden Beispielen sind zwei der Funktionen des Kommandos erkennbar:

Durch

```
man man
```

wird die Beschreibung des Kommandos *man* ausgegeben, wie sie im UNICOS User Commands Reference Manual SR-2011 enthalten ist. Dort findet sich auch eine vollständige Liste der verfügbaren online-Dokumentation der Fa. CRAY.

Durch

```
man -k string
```

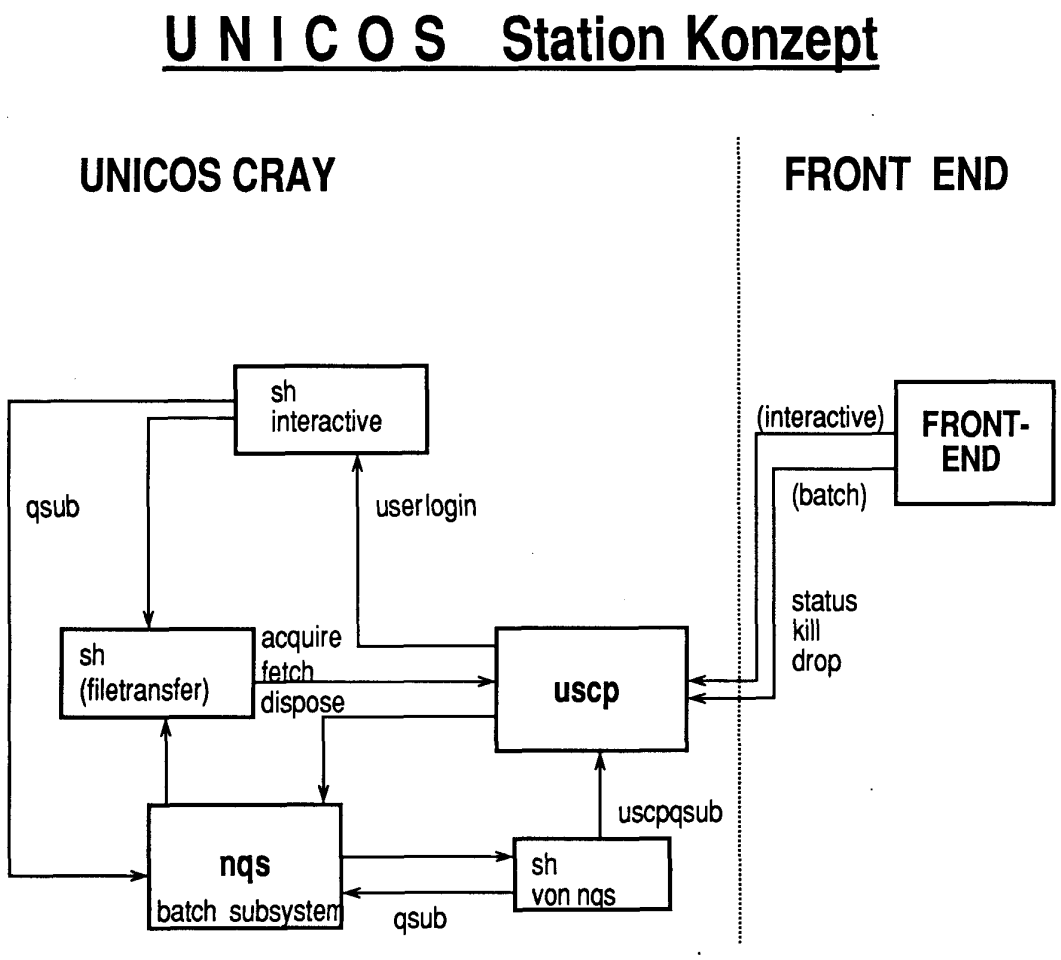
erhält man eine Liste von einzeiligen Kurzbeschreibungen, die in den verfügbaren Manuals unter dem Stichwort *NAME* aufgeführt sind und die Zeichenfolge *string* enthalten.

Hinweis: Wenn die von *man* erzeugte Beschreibung auf einen abgesetzten Rechner ohne UNIX-Betriebssystem erfolgen soll, sind zur Vermeidung von Ersatzdarstellungen für Fettdruck, Kursiv etc. zusätzliche Optionen empfehlenswert, z.B.

```
man -oul man
```

5. Zugang zur CRAY über CD-Rechner

Die Einbindung der CRAY in Rechnernetze ist im Kapitel 1.4 beschrieben. Auf den Vorrechnern arbeiten Software-Produkte, die sogenannten *Stations*, die Stapelaufträge entgegen nehmen, Listen und Mitteilungen liefern und den Austausch von Dateien vornehmen. Die jeweilige *Station* ist dem Wirtssystem angepaßt: Unter NOS/BE z.B. kommuniziert sie unmittelbar mit den Eingabe- und Ausgabewarteschlangen dieses Betriebssystems. Unter NOS/VE werden viele nützliche Eigenschaften unterstützt, z.B. Namens- und Abkürzungsregelungen, Einstieg in andere Umgebungen, Full-Screen-Modus etc. Die *Station* auf dem jeweiligen Vorrechner kommuniziert mit dem USCP (UNICOS Station Call Processor, vergl. Bild 'UNICOS Station Konzept'). USCP überprüft die Benutzervalidierung mit Hilfe der ersten Anweisung (#USER...) und übermittelt den Job nach erfolgreicher Prüfung dem Network Queueing System NQS zur weiteren Bearbeitung (siehe Kapitel 3.2).



5.1 Abschicken von Jobs an die CRAY

5.1.1 Abschicken von Jobs an die CRAY vom Vorrechner direkt

Das Abschicken von Jobs zur CRAY geschieht unter **NOS/VE** mit dem Kommando

SUBMIT_CRAY_JOB (alias SUBCJ, SUBMIT, SUBM, SUB):

```
SUBMIT_CRAY_JOB
FILE, F = filename
OUTPUT_DISPOSITION, STANDARD_OUTPUT, SO, ODI, OD = file or key
SUBSTITUTION_MARK, SM = 1 character string
USER_JOB_NAME, UJN, JN = jobname
```

und unter **NOS/BE** mit dem Kommando

CSUBMIT (alias CSUB):

CSUB[MIT],lfn,JN = jobname,TI = tid

Parameterbeschreibung:

FILE lfn		Name der Datei, die den CRAY-Job enthält.
OUTPUT_DISPOSITION: (nur NOS/VE)	DISCARD_ALL_OUTPUT DAO	Alle Ausgabedateien, auch die mit DISPOSE erzeugten, werden am Jobende gelöscht.
	DISCARD_STANDARD_OUTPUT DSO	Die UNICOS-Standard-Ausgabedatei *stdout* wird am Jobende gelöscht. Die mit DISPOSE erzeugten Ausgabedateien bleiben erhalten.
	LOCAL L	Druckt alle Ausgabedateien an der CYBER 930, auch wenn der Job von einem anderen Rechner abgeschickt wurde.
	PRINTER P	Einreihen in die NOS/VE-Print-Queue in Abhängigkeit von den Job-Attributen OUTPUT_DESTINATION und FORMS_CODE. (Die benutzereigenen Default-Werte erfährt man mit DISPLAY_JOB_ATTRIBUTES; die Veränderung dieser Werte erfolgt mit CHANGE_JOB_ATTRIBUTES).
	WAIT_QUEUE WQ WT	(Default): Bedeutet, daß die Ausgabedatei im Katalog $USER.$WAIT_QUEUE abgelegt wird. Als Dateiname wird der Wert des Parameters FILE bzw., wenn angegeben, USER_JOB_NAME verwendet.
SUBSTITUTION_MARK (nur NOS/VE)		Begrenzungszeichen für einen zu substituierenden String. Dieser Wert kann bei Verwendung des CRAY-Environment-Prolog (ENTCE, siehe Kapitel 5.4), durch die Systemvariable CRV$SUBSTITUTION_MARK vorbesetzt werden.

USER_JOB_NAME JN	Jobname, unter dem der CRAY-Job und dessen Ausgabe identifiziert wird. Erfolgt beim Aufruf von NOS/VE keine Angabe, so wird als Dateiname der Pfadname des File-Parameters verwendet. Unter NOS/BE bleibt die Angabe eines Jobnamens ohne Wirkung, da der CRAY-Job denselben Namen wie der NOS/BE-Job erhält.
TI (nur NOS/BE)	NOS/BE Terminal-Identifikation für die Ausgabe. Wird keine Angabe gemacht, wird die Terminal-Identifikation des absendenden Jobs dem CRAY-Job mitgegeben.

Anmerkung für NOS/BE: Eingabe und Ausgabe erfolgen im ASCII-95-Code; der Display-Code ist insbesonders für die Eingabe nicht zulässig! PRO ist nicht in der Lage, ASCII-95-Output zu verarbeiten; dieser muß durch *BATCH,lfn,LOCAL* in ein lokales File umgewandelt werden und kann dann z.B. mit *PAGE,lfn* und *L,73* angesprochen werden.

Beispiel: In der Datei *FIRST* sei der folgende UNICOS-Job abgelegt (siehe auch Seite 3-1):

```
# USER=userid PW=password
# QSUB-r first    #Jobname
# QSUB-lm 1000Kw
# QSUB-lT 2:30
cd $TMPDIR
cat << 'ZZ' > first.f
      write(*,*) ' hello world'
      end
ZZ
cft77 -es first.f
cat first.l
segldr first.o
a.out
exit
```

Mit den folgenden Kommandos wird der Job zur CRAY geschickt:

NOS/VE: `SUBMIT_CRAY_JOB FIRST` (abgekürzt: `SUBCJ FIRST`)

NOS/BE: `CSUBMIT,FIRST.`

Die Ausgabeliste wird wie folgt abgelegt:

NOS/VE: `$USER.$WAIT_QUEUE.FIRST`

(anzuschauen z.B. mit `COPF $USER.$WAIT_QUEUE.FIRST`)

NOS/BE: (als *Remote Output File*; aufgelistet mit Hilfe des Kommandos *FILES*, ansehen mit:)

```
BATCH,name,LOCAL
PAGE,name
L,1
```

Beim Zugang über die CYBER 930 (NOS/VE) besteht die Möglichkeit, daß spezielle Zeichenketten der Jobdatei, z.B. das Jobpaßwort, von der CDC NOS/VE Link Software (Command Reference Manual: SC-0270 B) während des SUBMIT-Kommandos erfragt und eingesetzt werden.

Beispiel: Die erste Zeile des Jobs aus der Datei FIRST laute

```
# USER=userid PW=&$PP('password, please: ',true)&
```

Nach Aufruf mit

```
SUBCJ FIRST SM = '&'
```

fragt das Kommando nach dem Paßwort:

```
password, please:
```

Der eingegebene String wird wegen der Angabe des *secure_parameters* "true" nicht als Echo an das Terminal zurückgeschickt, und der Bereich auf dem Bildschirm wird nach Betätigen der ENTER-Taste gelöscht. Mit dem am Bildschirm eingegebene String wird die im Job durch &... & begrenzte Zeichenfolge ersetzt; dann wird der Job an die CRAY geschickt.

5.1.2 Abschicken von Jobs an die CRAY von abgesetzten NOS/VE Anlagen über QTF

Auf der NOS/VE-Anlage der FUB ist eine Untermenge der CRAY-Station-Software installiert, die den Transfer von CRAY-Jobs und deren Ausgabe initiiert und überwacht. Der Job- bzw. Output-Transfer arbeitet mit dem CD-Produkt QTF (*Queuefile Transfer Facility*, siehe Kapitel 1.4.5), ohne daß der Benutzer selbst QTF aufrufen muß. Dieser Weg ist unabhängig von der DFN-Software. Der Job muß vollständig mit allen Parametern versehen mit dem folgenden Kommando abgeschickt werden:

```
SUBMIT_CRAY_JOB   (alias SUBCJ)
```

Es gelten die gleichen Parameter wie in 5.1.1 beschrieben. Ohne Angabe von Parametern gelangt die Druckausgabe des CRAY-Jobs über den Vorrechner CYBER 930 des ZIB zum abgesetzten NOS/VE-Rechner zurück. Der Standard-Katalog ist $WAIT_QUEUE, als Dateiname wird der Wert des Parameters FILE bzw., wenn angegeben, USER_JOB_NAME verwendet. Ein Dateitransfer von der CRAY aus, mittels *acquire*, *fetch* oder *dispose*, bezieht sich stets auf den im CDCNET eingebundenen Vorrechner CYBER 930 des ZIB. Es ist eine vollständige Validierung des Benutzers erforderlich, z.B.:

```
fetch file -t 'LU = BF01XXXX  LA = FUB PW = XXXX'.
```

Den gültigen Schlüsselworten sind die eigenen aktuellen Account-Parameter zuzuordnen.

5.1.3 Abschicken von Jobs an die CRAY mit Hilfe des RJE-Dienstes des DFN über NOS/VE

Über den auf dem Vorrechner CYBER 930 unter NOS/VE installierten DFN-RJE-Dienst kann die CRAY X-MP des ZIB auch direkt erreicht werden. Die zugehörige DFN-Adresse (PIX-Name) lautet:

```
B   ZIB12
```

Eine Beschreibung dieses Dienstes erhält man mit

```
DOC,DFN,RJEVE.
```

Beispiele für Jobs zur CRAY X-MP des ZIB:

Jobtransfer von der CYBER 860 (NOS/BE) der ZRZ der TU Berlin zur CRAY X-MP des ZIB

Der folgende Batchjob, der im File *CRYJOB* an der CYBER 860 (NOS/BE) der ZRZ im ASCII-Code gespeichert ist, soll an der CRAY X-MP des ZIB ausgeführt werden:

```
# USER=userid PW=password
# QSUB-r first
# QSUB-lm 1000Kw
# QSUB-lT 2:30
cd $TMPDIR
cat << '%%' > first.f
           write(*,*) ' hello world'
           end
%%
cft77 -es first.f
cat first.l
segldr first.o
a.out
exit
```

Der Aufruf ist dann gemäß RJE-Implementierung auf NOS/BE-Anlagen

```
RJE,CRYJOB,D = DIR
```

wobei das Direktivenfile *DIR* (nur im Display-Code möglich) die für den Jobtransfer nötigen Angaben bezüglich *destination host* und Validierung auf der CYBER 930 (NOS/VE) enthält (Leerzeichen in der *destination host id* sind signifikat!):

```
DHID=B  ZIB12,veuserid,,vepassword
```

Die Job-Ausgabe wird nach Ablauf des Jobs (im ASCII-Code) auf der NOS/BE-Anlage in die Output-Queue des aufrufenden Benutzers gelegt, d.h. sie kann mit

```
BATCH,Ammmmnn,LOCAL
```

aus der Queue in ein lokales File geschrieben und dann weiterverarbeitet werden, z.B. mit

```
PAGE,Ammmmnn
L,1
```

Anmerkungen:

* CRAY-Jobs müssen ASCII-codiert sein, DISPLAY-Code ist nicht zulässig!
* Eine Beschreibung der RJE-Implementierung für Rechner mit dem Betriebssystem NOS/BE ist unter DOC,BERNET,RJE erhältlich.

Jobtransfer von der VAX (VMS) der BAM zur CRAY X-MP des ZIB:

Im File *CRYJOBJOB* der VAX (VMS) soll derselbe CRAY-Job wie im vorigen Beispiel gespeichert sein. Der Aufruf erfolgt dann gemäß DFN-RJE-Implementierung auf der VMS-Anlage mit

```
RJE/JOB CRYJOB.JOB "B  ZIB12"/UEX=(veuserid,"X",vepassword) -
"B  BAM03"/UOUT=(vaxuser,"X",vaxpw) -
/ODS=[DIRECTORY]FILENAME.OUT
```

Die Job-Ausgabe wird zurück zur BAM geschickt und in die Datei [DIRECTORY]FILENAME.OUT abgelegt.

Anmerkung:

- Eine ausführliche Beschreibung des DFN-RJE-Dienstes auf der VAX (VMS) enthält das Benutzerhandbuch für den DFN-Basisdienst "REMOTE JOB ENTRY auf Anlagen des Typs VAX unter VMS" der Firma Digital Equipment GmbH.

Anmerkungen zu beiden Beispielen:

- Die zur Validierung auf der CRAY nötigen Angaben werden in Kleinbuchstaben im CRAY-Job selbst angegeben und nicht im Direktivenfile.

- Wenn im CRAY-Job Bezug auf den Vorrechner genommen wird (z.B. *fetch*, *dispose*), muß im Direktivenfile die Validierung für den NOS/VE-Vorrechner nach der Destination-Host ID stehen (DHID = B ZIB12,veuserid,,vepassword).

- Wenn im CRAY-Job kein Bezug auf den Vorrechner genommen wird, kann die Angabe *veuserid,,vepassword* entfallen. Benötigt die Implementation der DFN-Software auf dem absendenden Host Angaben für *veuserid,,vepassword*, so darf als der Inhalt von Parameter *veuserid* DFNCRAY angegeben werden; das *vepassword* ist dann beliebig.

5.1.4 Abschicken von Jobs an die CRAY mit Hilfe des RJE-Dienstes des DFN über NOS/BE

Auch über den auf dem Vorrechner CYBER 825 unter NOS/BE installierten DFN-Dienst kann die CRAY X-MP des ZIB direkt erreicht werden. Als DFN-Adresse muß in diesem Fall jedoch die der CYBER 825 verwendet werden, also

```
B  ZIB01
```

Der CRAY-Job (vgl. Beispiel von Kapitel 5.1.2) muß vorn um eine Anweisung (auch im ASCII-Code) für die CYBER 825 ergänzt werden (dabei ist *nnnn* die vierstellige Auftragsnummer des Benutzers an der CYBER 825):

```
Annnn,STUNI.
```

Beispiel:

Jobtransfer von der CYBER 860 (NOS/BE) der ZRZ der TU Berlin zur CRAY X-MP des ZIB

Der folgende Batchjob, der im File *CRYJOB* an der CYBER 860 (NOS/BE) der ZRZ im ASCII-Code gespeichert ist, soll an der CRAY X-MP des ZIB ausgeführt werden:

```
A2345,STUNI.
# USER=userid PW=password
# QSUB-r first
# QSUB-lm 1000Kw
# QSUB-lT 2:30
cd $TMPDIR
cat << 'ZZ' > first.f
      write(*,*) ' hello world'
      end
ZZ
```

```
cft77 -es first.f
cat first.l
segldr first.o
a.out
exit
```

Zum Absenden genügt der Aufruf:

```
RJE,CRYJOB.
```

Die Job-Ausgabe wird nach Ablauf des Jobs (im ASCII-Code) auf der absendenden NOS/BE-Anlage in die Output-Queue des aufrufenden Benutzers gelegt, d.h. er kann mit

```
BATCH,Ammmmnn,LOCAL
```

aus der Queue in ein lokales File geschrieben und dann weiterverarbeitet werden, etwa mit

```
PAGE,Ammmmnn
L,1
```

Anmerkungen:

- CRAY-Jobs müssen ASCII-codiert sein, DISPLAY-Code ist nicht zulässig!

- Eine Beschreibung der RJE-Implementierung für Rechner mit dem Betriebssystem NOS/BE ist unter DOC,BERNET,RJE erhältlich.

5.2 Kommandos zum Dateitransfer zwischen CRAY und Vorrechner

Die Plattenlaufwerke der CRAY dienen der schnellen Ein-/Ausgabe und nicht der langfristigen Datenhaltung. Daher sollte langfristige Datenhaltung ausschließlich auf den Vorrechnern erfolgen.

5.2.1 Dateitransfer zur CRAY

Mit den Standard-UNICOS-Kommandos *fetch* und *acquire* sowie dem ZIB-Kommando *getbe* werden Dateien zur CRAY übertragen. Das Kommando *fetch* fordert auf dem gewählten Vorrechner die gewünschte Datei an. Eine auf der CRAY eventuell existierende Datei gleichen Namens wird überschrieben. Das *fetch*-Kommando wartet bis zum erfolgreichen Abschluß der Übertragung. Das Kommando *acquire* arbeitet ähnlich wie *fetch*, prüft jedoch zunächst, ob auf der CRAY eine Datei des angeforderten Namens schon existiert und überträgt diese Datei vom Vorrechner zwar wie *fetch*, jedoch nur, wenn sie auf der CRAY nicht vorhanden war.

Standard-UNICOS-Kommandos *fetch* und *acquire*

```
fetch localpath -n sfn -m mf -d dc -f fm -t 'text'

acquire localpath -n sfn -m mf -d dc -f fm -t 'text'
```

Parameter (die Voreinstellung (Default) ist unterstrichen):

localpath *pathname:* UNICOS-Dateiname.

-n sfn
localpath
: *source file name*: Dateiname, unter dem die Datei auf dem Vorrechner angesprochen wird (maximal 15 Zeichen lang); bei Verwendung der NOS/VE-Station wird eine Datei in UNICOS-Schreibweise benannt, also z.B. *dir/file.f* anstelle *DIR.FILE_F*.
Bestimmte Sonderzeichen in NOS/VE-Dateinamen werden auf andere Sonderzeichen abgebildet:
/ wird zu .
. wird zu _
(Siehe auch CDC NOS/VE Link Software Command Reference Manual SC-0270 B, Kapitel 2.9).

-m mf
: *Mainframe*: Bezeichnung des Vorrechners, von dem die Datei übertragen wird (2 Großbuchstaben). Voreinstellung für *mf* ist der Vorrechner, von dem der Job zur Cray geschickt wurde:

VE
: CYBER 930 (NOS/VE).

BE
: CYBER 825 (NOS/BE).

Disposition Code:

-d dc

ST
: Permanente Datei auf dem Vorrechner.

MT
: (nur NOS/VE:) Magnetband auf dem Vorrechner.

-f fm
: *Dataset Format*: Dieser Parameter bestimmt die Übertragung der Struktur und des Zeichensatzes:

BB
: *Binary Blocked*: Vorrechnerstruktur -> UNICOS-Struktur.

TR
: *Transparent*: Übertragung als Bitfolge.

TB
: *Transparent Binary Blocked*: Vorrechnerstruktur -> COS-Blocked-Struktur.

TC
: *Transparent Character Blocked*: Vorrechnerstruktur -> COS-Block-Struktur.

UD
: *UNIX-Data*: Vorrechner-Struktur -> UNICOS-Struktur, ASCII-Vorrechner-Zeichensatz -> ASCII-Code (NOS/BE: 8/12 bit ASCII, auch ASCII95 oder ASCII8 genannt).

-t 'text'
: *'text'* enthält weitere Kommandos für den Transfer vom Vorrechner. Für die CRAY ist *'text'* nur eine Zeichenkette, die nicht weiter interpretiert wird, jedoch an den Vorrechner zur Ausführung übertragen wird. Die Syntax dieser Zeichenkette ist deshalb in den entsprechenden Abschnitten (NOS/VE - 5.4.1, NOS/BE - 5.5) aufgeführt.

Die Anweisung *fetch* ist insbesondere dann der Anweisung *acquire* vorzuziehen, wenn man auf jeden Fall auf die am Vorrechner gespeicherte Datei zugreifen will, z.B. weil sie eventuell einen aktuelleren Stand als die an der CRAY noch vorhandene Datei gleichen Namens hat. Die Anweisung *acquire* sollte verwendet werden, wenn an einem Tag mehrere Programmläufe mit dieser Datei (in unveränderter Version) vorgesehen sind. Man beachte, daß das jeweilige Übertragen der Datei mehr Zeit kostet als der Zugriff auf eine Datei an der CRAY.

Anmerkung: Um einen Dateitransfer zwischen CRAY und Vorrechner auszuführen, muß auf dem jeweiligen Vorrechner ein Job gestartet werden - d.h., alle Prologe auf dem Vorrechner müssen abgearbeitet werden; danach wird der Datentransport vorgenommen. In einer Sekunde können transparent bei NOS/VE maximal 8 Mbit, bei NOS/BE maximal 3 Mbit (zum Vergleich: 1 Mbit entspricht etwa 30 Sektoren bzw. 250 PRU's) oder codiert bei NOS/VE maximal 3 Mbit, bei NOS/BE maximal 700 Kbit übertragen werden. Sofern die Dateien auf dem Vorrechner nicht bearbeitet werden müssen, sollte man darauf achten, daß bei den Kommandos *acquire*, *dispose* und *fetch -fTR* angegeben wird, da die Voreinstellung *-fUD* ist.

Beispiele:

```
fetch prog
```

Dieses Kommando lädt die NOS/VE-Datei *$USER.PROG* im ASCII-Code von der NOS/VE-Anlage des ZIB nach *prog* im aktuellen Verzeichnis, sofern der Job von NOS/VE aus gestartet wurde.

```
acquire $HOME/problem1/prog.f -n problem1/prog.f -f TR
```

Dieses Kommando lädt transparent die NOS/VE-Datei *problem1.prog_f* von der NOS/VE-Anlage des ZIB nach *$HOME/problem1/prog.f*, sofern der Job von NOS/VE aus gestartet wurde.

Weitere Beispiele, insbesondere für NOS/BE, findet man im Anschluß an die Erläuterung des Text-Parameters im Kapitel 5.5.

Das Kommando *getbe* des ZIB

Das ZIB-Kommando *getbe* (get from NOS/BE-Front-End: Datei vom NOS/BE Vorrechner übertragen) ist ein vom ZIB geschaffenes bequemeres Werkzeug als die Kommandos *fetch* und *acquire* im Zusammenspiel mit dem NOS/BE-Vorrechner.

Der Benutzer spart die mühsame und fehleranfällige Erstellung der transferspezifischen Angaben (z.B. Erstellen eines vollständigen NOS/BE-Jobs) im Textfeld des Kommandos *acquire*. Das Kommando *getbe* lädt eine Datei aus einer NOS/BE-Family; es arbeitet ähnlich wie *acquire*, d.h. *getbe* stellt dem CRAY-Job eine Datei zur Verfügung. Existiert die Datei bereits als permanente Datei an der CRAY, wird diese angesprochen. Existiert die Datei nicht auf der CRAY, so wird sie vom Vorrechner übertragen, auf der CRAY permanent gemacht und dem Job zur Verfügung gestellt. Das Kommando *getbe* bricht den Prozess ab, wenn auf die Datei vom Vorrechner nicht zugegriffen werden kann.

Allgemeiner Aufruf von *getbe*

```
getbe localpath remotefile -f fm -l -n -u beuser -a beaccount -j bejobpw -F befamily -i beid\
      -t betkpw -r berdpw -x bexrpw
```

Parameter:

localpath	Pfadname: UNICOS-Dateiname. Dieser Parameter ist Pflicht.
remotefile	*Internal Filename*: Indirekter Filename einer BE-Family. Dieser Parameter ist Pflicht.
-f fm	*Dataset Format* (vgl. *fetch*). Zusätzlich gibt es den Parameterwert DI, der auf dem
TR	NOS/BE-Vorrechner die Datei von Display-Code in ASCII-Code umwandelt und danach im Format UD überträgt. Default: TR (abweichend von *fetch*).
-l	Das *dayfile* des BE-Jobs wird nach *stderr* kopiert. Default: das *dayfile* des BE-Jobs wird nur im Fehlerfall nach *stderr* kopiert.
-n	Die Datei wird bedingungslos von der CYBER 825 geholt (*fetch*). Default: die Datei wird nur dann von der CYBER 825 geholt, wenn sie noch nicht vorhanden ist (*acquire*).
-u beuser	*BE User*. Default: Inhalt der Variablen BE_USER.
-a beaccount	*BE Account*. Default: Inhalt der Variablen BE_ACCOUNT.
-j bejobpw	*BE-Jobpassword*. Default: Inhalt der Variablen BE_JOBPW.
-F befamily	*BE-Family*. Default: Inhalt der Variablen BE_FAMILY. Ist BE_FAMILY leer, so gilt der Wert: CRAYFAM.
-i beid	*BE Permfile Identification*. Default: Inhalt der Variablen BE_ID. Ist BE_ID leer, so ist der Default der Inhalt von BE_USER.
-t betkpw	*BE Turnkey Password*. Default: Inhalt der Variablen BE_TK.
-r berdpw	*BE Read Password*. Default: Inhalt der Variablen BE_RD.
-x bexrpw	*BE XR Password*. Default: Inhalt der Variablen BE_XR.
	Sind keine Family-Paßworte gesetzt, so wird das *Turnkey Password* auf den Inhalt von BE_JOBPW gesetzt.

Einige Parameter können auch als Systemvariable übergeben werden (insbesondere für Paßworte):

Beispiel:

```
BE_USER=4711; export BE_USER
BE_FAMILY=crayfamily; export BE_FAMILY
set +vx         # kein Auflisten der Passwoerter nach stdout
BE_JOBPW=geheim; export BE_JOBPW
BE_TK=tkpass; export BE_TK
BE_XR=xrpass; export BE_XR
BE_RD=rdpass; export BE_RD
set -vx         # alte Werte wieder setzen

getbe affe test
```

Die Datei *TEST* in der NOS/BE Family *CRAYFAMILY, ID = 4711* wird, falls die UNICOS-Datei *affe* in dem voreingestellten Verzeichnis nicht existiert, transparent übertragen und unter *affe* bereitgestellt.

```
getbe tst tst -n
```

Die Datei *TST* in der NOS/BE-Family *CRAYFAMILY, ID = 4711* wird in jedem Fall in die UNICOS-Datei *tst* des voreingestellten Verzeichnisses kopiert.

5.2.2 Dateitransfer zum Vorrechner

Dateien, die auf der CRAY bearbeitet wurden (z.B. Ausgabelisten, Programmdaten oder Jobs) können zu einem der Vorrechner übertragen werden. Auf diese Weise können einerseits CRAY-Dateien beim Vorrechner permanent gespeichert werden, Ausgabelisten gedruckt oder über DFN weitergeleitet werden, andererseits können Jobs in die Eingabe-Warteschlange des Vorrechners eingereiht werden. Die Übertragung dieser Dateien erfolgt mit den Kommandos *dispose* oder *putbe*.

Das Standard-UNICOS-Kommando *dispose* ist flexibel und für alle Vorrechner anwendbar. Der Benutzer stellt im Parameter *-t* alle Transferinformationen bereit. Das Kommando *putbe* dient zum Ablegen der Datei in eine Family an der CYBER 825 (analoges Kommando zu *getbe*).

Das Standard-UNICOS-Kommando *dispose*

Das Standard-UNICOS-Kommando *dispose* dient zum flexiblen Übertragen der Dateien von der CRAY zum Vorrechner.

```
dispose localpath -n sfn -m mf -d dc -f fm -t text
```

Parameter (die Voreinstellung (Default) ist unterstrichen):

localpath Pfadname: UNICOS-Dateiname.

-n sfn localpath	*source file name*: Dateiname, unter dem die Datei auf dem Vorrechner angelegt wird (maximal 15 Zeichen lang); bei Verwendung der NOS/VE-Station wird eine Datei in UNICOS-Schreibweise benannt, also z.B. *dir/file.f* anstelle *DIR.FILE_F*. Bestimmte Sonderzeichen in NOS/VE-Dateinamen werden auf andere Sonderzeichen abgebildet: / wird zu . . wird zu _ (Siehe auch CDC NOS/VE Link Software Command Reference Manual SC-0270 B, Kapitel 2.9)
-m mf	Mainframe: Bezeichnung des Vorrechners, zu dem die Datei übertragen wird (2 Großbuchstaben). Voreinstellung für *mf* ist der Vorrechner, von dem der Job zur Cray geschickt wurde.
VE	CYBER 930 (NOS/VE)
BE	CYBER 825 (NOS/BE)
-d dc	Disposition Code:
ST	Permanente Datei auf dem Vorrechner.
MT	Magnetband auf dem Vorrechner.
PR	Druckausgabe auf dem Vorrechner.
IN	Eingabejob auf dem Vorrechner.
-f fm	*Dataset Format*: Dieser Parameter bestimmt die Übertragung der Struktur und des Zeichensatzes.
BB	*Binary Blocked*: UNICOS-Struktur -> Vorrechnerstruktur.
TR	*Transparent*: Übertragung als Bitfolge.
TB	*Transparent Binary Blocked*: COS-Blocked-Struktur -> Vorrechnerstruktur.
TC	*Transparent Character Blocked*: COS-Block-Struktur -> Vorrechnerstruktur.
UD	*UNIX-Data*: Vorrechner-Struktur -> UNICOS-Struktur, ASCII-Vorrechner-Zeichensatz -> ASCII-Code (NOS/BE: 8/12 bit ASCII, auch ASCII95 oder ASCII8 genannt).
-t 'text'	*'text'* enthält weitere Kommandos für den Transfer zum Vorrechner. Für die CRAY ist 'text' nur eine Zeichenkette, die nicht weiter interpretiert wird, jedoch an den Vorrechner zur Ausführung übertragen wird. Die Syntax dieser Zeichenkette ist deshalb in den entsprechenden Abschnitten (NOS/VE - 5.4.2, NOS/BE - 5.5) aufgeführt.

Anmerkung: In einer Sekunde können transparent bei NOS/VE maximal 5 Mbit, bei NOS/BE maximal 2 Mbit (zum Vergleich: 1 Mbit entspricht etwa 30 Sektoren bzw. 250 PRU's) oder codiert bei NOS/VE maximal 1,5 Mbit, bei NOS/BE maximal 200 Kbit übertragen werden.

Beispiele:

```
dispose prog
```

Dieses Kommando sendet die UNICOS-Datei *prog* im aktuellen Verzeichnis in die Output-Queue der NOS/VE-Anlage des ZIB, sofern der Job von NOS/VE aus gestartet wurde.

```
dispose $HOME/problem1/prog.f -d ST -t 'f=problem1.prog_f' -f TR
```

Dieses Kommando speichert die UNICOS-Datei *$HOME/problem1/prog.f* transparent auf die NOS/VE-Anlage des ZIB unter dem Namen *$USER.PROBLEM1.PROG_F*, sofern der Job von NOS/VE aus gestartet wurde.

Weitere Beispiele, insbesondere für NOS/BE, findet man nach Erläuterung des Text-Parameters Kapitel 5.5.

Das Kommando *putbe* des ZIB

Das ZIB-Kommando *putbe* (put to NOS/BE-Front-End: Datei auf dem NOS/BE-Vorrechner speichern) sichert eine Datei auf dem NOS/BE-Vorrechner CYBER 825 in eine NOS/BE-Family; es ist ein vom ZIB geschaffenes bequemeres Werkzeug als *dispose* im Zusammenspiel mit dem NOS/BE-Vorrechner: der Benutzer spart die mühsame und fehleranfällige Erstellung der transferspezifischen Angaben (z.B. Erstellen eines vollständigen NOS/BE-Jobs) im *dispose*-Textfeld.

Allgemeiner Aufruf von *putbe*

```
putbe localpath remotefile -f fm -l -u beuser -a beaccount -j bejobpw -F befamily -i beid\
      -t betkpw -r berdpw -x bexrpw
```

Parameter:

localpath	Pfadname: UNICOS-Dateiname. Dieser Parameter ist Pflicht.
remotefile	File einer BE-Family. Dieser Parameter ist Pflicht.
-f fm	*Dataset Format* (vgl. *dispose*). Zusätzlich gibt es den Parameterwert DI, der die
TR	UNICOS-Datei im Format UD überträgt und danach auf dem NOS/BE-Vorrechner von ASCII-Code in Display-Code umwandelt. Default: TR (abweichend von *dispose*).
-l	Das *dayfile* des BE-Jobs wird nach *stderr* kopiert. Default: das *dayfile* des BE-Jobs wird nur im Fehlerfall nach *stderr* kopiert.
-u beuser	*BE User*. Default: Inhalt der Variablen BE_USER.
-a beaccount	*BE Account*. Default: Inhalt der Variablen BE_ACCOUNT.
-j bejobpw	*BE-Jobpassword*. Default: Inhalt der Variablen BE_JOBPW.
-F befamily	*BE-Family*. Default: Inhalt der Variablen BE_FAMILY. Ist BE_FAMILY leer, so gilt der Wert: CRAYFAM.
-i beid	*BE Permfile Identification*. Default: Inhalt der Variablen BE_ID. Ist BE_ID leer, so ist der Default der Inhalt von BE_USER.
-t betkpw	*BE Turnkey Password*. Default: Inhalt der Variablen BE_TK.
-r berdpw	*BE Read Password*. Default: Inhalt der Variablen BE_RD.
-x bexrpw	*BE XR Password*. Default: Inhalt der Variablen BE_XR.
	Sind keine Family-Paßworte gesetzt, so wird das *Turnkey Password* auf den Inhalt von BE_JOBPW gesetzt.

Einige Parameter können auch als Systemvariable übergeben werden (insbesondere für Paßworte).

Beispiel:

```
BE_USER=4711; export BE_USER
BE_FAMILY=crayfamily; export BE_FAMILY
set +vx         # kein Auflisten der Passwörter nach stdout
BE_JOBPW=geheim; export BE_JOBPW
BE_TK=tkpass; export BE_TK
BE_XR=xrpass; export BE_XR
BE_RD=rdpass; export BE_RD
set -vx         # alte Werte wieder setzen

putbe affe test
```

Die UNICOS-Datei *affe* wird transparent übertragen und in der NOS/BE-Family *CRAYFAMILY,ID = 4711* abgelegt.

5.2.3 Dateitransfer mit einer abgesetzten NOS/VE-Anlage

Um eine Datei von oder zu einer über PTF (*Permfile Transfer Facility*, siehe Kapitel 1.4.5) erreichbaren NOS/VE-Anlage zu übertragen, wird einerseits die Information zur Validierung an dieser Anlage benötigt und andererseits der Name des Host, von oder zu dem die Datei übertragenden werden soll. Die Angabe dieser Daten hängt davon ab, ob der zugehörige Job vom Vorrechner der CRAY oder von einer anderen NOS/VE-Anlage über QTF (*Queuefile Transfer Facility*, siehe Kapitel 1.4.5) zur CRAY geschickt wird.

Abschicken des Jobs an die CRAY direkt vom Vorrechner

Das Abschicken des Jobs erfolgt wie unter 5.1.1 beschrieben. Zuvor muß jedoch mit dem Kommando CREATE_STATION_VALIDATION die Validierung für die abgesetzte NOS/VE-Anlage gegeben werden.

CREATE_STATION_VALIDATION (alias CRESV)
HOST, H, LOCATION, L = name or string or key
VALIDATION, V = string

Parameter:

HOST : ATTACHED	Dieser Schlüssel gibt an, daß die im Parameter *validation* angegebene Information für den Vorrechner der CRAY gelten soll.
FUB	Gibt an, daß die im Parameter *validation* angegebene Information für die NOS/VE-Anlage der ZEDAT der FUB gelten soll.
VALIDATION	Angabe der Validierung für die unter Parameter *Host* angegebene NOS/VE-Anlage.

Beispiel für eine permanente Festlegung der Validierung:

```
CREATE_STATION_VALIDATION HOST=FUB VALIDATION='LOGIN LU=FREMD PW=GEHEIM'
```

Zum befristeten Festlegen der NOS/VE-Anlage, von oder zu der Dateien mit den UNICOS-Kommandos *acquire*, *fetch* oder *dispose* transportiert werden, gibt es folgende Möglichkeiten:

* Mit dem Kommando CHANGE_FILE_TRANSFER_HOST (alias CHAFTH) wird die angegebene NOS/VE-Anlage zu dem Host erklärt, von oder zu dem die Dateien übertragen werden.

  CHANGE_FILE_TRANSFER_HOST (alias CHAFTH)
  HOST,H = name or string or key

  Parameter:

HOST : ATTACHED	Dieser Schlüssel gibt an, daß die Dateien vom oder zum Vorrechner übertragen werden.
FUB	Gibt an, daß die Dateien von oder zur NOS/VE-Anlage der ZEDAT der FUB übertragen werden.

* Bei den UNICOS-Kommandos *acquire*, *fetch* oder *dispose* wird im Textfeld angegeben, von oder zu welcher NOS/VE-Anlage die Datei übertragen wird.

  -t' ... HOST = FUB ... ' oder

  -t'f = :FUB.userid.dateiname'

Mit dem Kommando DISPLAY_STATION_VALIDATION (alias DISSV) kann man sich informieren, für welche NOS/VE-Anlagen Validierungsinformationen existieren.

Das Kommando DELETE_STATION_VALIDATION (alias DELSV) dient dem Löschen der Validierungsinformationen für eine NOS/VE-Anlage.

Parameter:

HOST : ALL	Es werden alle Validierungsinformationen für alle NOS/VE-Anlagen gelöscht.
FUB	Es wird die Validierungsinformationen für die NOS/VE-Anlage der ZEDAT der FUB gelöscht.

Abschicken von Jobs an die CRAY von abgesetzten NOS/VE-Anlagen über QTF

Das Abschicken von Jobs erfolgt wie unter 5.1.2 beschrieben. Die Kommandos CREATE_STATION_VALIDATION und CHANGE_FILE_TRANSFER_HOST müssen in eine Datei mit Namen $USER.STATION_VALIDATION an dieser Anlage eingetragen werden.

5.3 Jobinformation und Jobsteuerung

Der Benutzer kann über die Vorrechner die Gesamtheit der Jobs auf der CRAY beobachten (Status-Informationen), den Ablauf eigener Jobs verfolgen (Job-Information) und eigene Jobs manipulieren, z.B. vorzeitig beenden. Die jeweilige Kommandosyntax ist zwar vorrechnerabhängig, jedoch immer sehr ähnlich, deshalb werden diese Kommandos gemeinsam unabhängig vom Vorrechner aufgeführt. Diese Kommandos stehen nur auf den direkten Vorrechnern der CRAY zur Verfügung, da die Informationen jeweils aktuell abgerufen werden. Sollte zum Zeitpunkt des Aufrufs die Verbindung zur CRAY nicht aktiv sein, so können keine Informationen ausgegeben werden und das Kommando wird mit einer entsprechenden Meldung beendet.

5.3.1 Status-Information

Mit Hilfe der Programme DISPLAY_CRAY_STATUS (NOS/VE) bzw. CSTATUS (NOS/BE) kann sich der Benutzer über den aktuellen Status aller Jobs in den verschiedenen Warteschlangen informieren.

NOS/VE: DISPLAY_CRAY_STATUS (alias DISCS)

```
DISPLAY_CRAY_STATUS
        STATION_IDENTIFIER, SI, ID, MF = name
        TERMINAL_IDENTIFIER, TID, TI = name
        REFRESH_INTERVAL, RI = integer
        OUTPUT,O = file
        STATUS = var of status
```

NOS/BE: CSTATUS,L = lfn,RT = rt.

Parameter:

STATION_IDENTIFIER (nur NOS/VE)	Zweistellige Vorrechnerkennung: VE CYBER 930 (NOS/VE) BE CYBER 825 (NOS/BE)

TERMINAL_IDENTIFIER (nur NOS/VE)	1 ... 8 stellige Terminalkennung, gleich der Benutzerkennung des abzusendenden NOS/VE-Jobs. Bei Jobs von NOS/BE: *ssstt*. Dabei ist *sss* eine dreistellige Bezeichnung des Herkunftsrechners und *tt* die zweistellige NOS/BE Terminal-Identifikation.
REFRESH_INTERVAL RT	Zeitintervall in Sekunden (0 ... 60), nach dem jeweils der Bildschirminhalt erneuert wird. Default ist 0, also kein Neuaufbau! Die Ausgabe auf dem Bildschirm erfolgt bei NOS/VE im *Full Screen Modus* (wie z.B. EDIT_FILE), was voraussetzt, daß dem System der verwendete Terminaltyp bekannt ist. Bei RI = 1 ... 60 muß das Kommando mit *F6* (Funktionstaste 6) beendet werden.
OUTPUT L	*Output-Filename*: Ausgabedatei. Default ist der Bildschirm.
STATUS (nur NOS/VE)	Statusvariable für Fehlererkennung.

5.3.2 Jobinformationen

Mit Hilfe der Programme DISPLAY_JOB_INFORMATION (NOS/VE) bzw. CJOB (NOS/BE) kann man sich auf den Vorrechnern des ZIB ausführlich über den Status von Jobs informieren, die unter der eigenen Kennung auf der CRAY laufen.

NOS/VE: DISPLAY_JOB_INFORMATION (DISJI)

JOB_SEQUENCE_NUMBER, JSQ = integer 1 ... 65535
REFRESH_INTERVAL, RI = integer 0 ... 60
OUTPUT,O = file
STATUS = var of status

NOS/BE: CJOB,jn,jsq, L = out,RT = rt

Parameter:

jn (nur NOS/BE)	Name des Jobs in Großbuchstaben.
JOB_SEQUENCE_NUMBER jsq	*Jobsequencenumber*: Nummer des Jobs, die man mit DISCS oder CSTAT erhalten kann (nicht mit RST).
REFRESH_INTERVAL RT	*Refresh Intervall*: Zeitintervall in Sekunden (0 ... 60), nach dem jeweils der Bildschirminhalt erneuert wird. Default ist rt = 0, also kein Neuaufbau! Die Ausgabe auf dem Bildschirm erfolgt im *Full Screen Modus* (wie z.B. EDIT_FILE), was voraussetzt, daß dem System der verwendete Terminaltyp bekannt ist. Bei RI = 1 ... 60 muß das Kommando mit *F6* beendet werden.
OUTPUT L	Ausgabedatei. Default: Bildschirm.

STATUS (nur NOS/VE)	Statusvariable für Fehlererkennung.

5.3.3 Abbrechen und Beenden von Jobs

Abbrechen von Jobs vom Vorrechner aus

Die Programme DROP_JOB (NOS/VE) bzw. CDROP (NOS/BE) erlauben dem Benutzer, vom Vorrechner aus seinen gerade in Ausführung befindlichen CRAY-Job abzubrechen. Die bisher erzeugte Ausgabe (*stdout*) geht verloren.

NOS/VE: DROP_JOB (alias DROJ)

JOB_SEQUENCE_NUMBER, JSQ = integer 1 ... 65535
STATUS

NOS/BE: CDROP,jsq.

Parameter:

JSQ jsq	Jobsequencenumber: Nummer des Jobs, die man mit DISCS oder CSTAT erhalten kann (nicht mit RST!).
STATUS (nur NOS/VE)	Statusvariable für Fehlererkennung.

Beenden von Jobs vom Vorrechner aus

Die Wirkung von KILL_JOB (NOS/VE) bzw. CKILL (NO/BE) hängt vom aktuellen Status des Jobs ab: ein Job in der Eingabe-Warteschlange wird aus dieser entfernt; ein Job in der Ausführungs-Warteschlange wird abgebrochen, der Benutzer erhält keine Ausgabe (Löschen von *stdout*); von einem Job in der Ausgabe-Warteschlange wird die gesamte Ausgabe gelöscht (Löschen von *stdout*).

NOS/VE: KILL_JOB (alias KILJ)

JOB_SEQUENCE_NUMBER, JSQ = integer 1 ... 65535
STATUS

NOS/BE: CKILL,jsq.

Parameter:

JSQ jsq	Jobsequencenumber: Nummer des Jobs, die man mit DISCS oder CSTAT erhalten kann; nicht mit RST!
STATUS (nur NOS/VE)	Statusvariable für Fehlererkennung

Hinweis: CRAY-Jobs können auch mit dem UNICOS-Kommanmdo *deljob* (siehe Kapitel 4.8) gelöscht werden.

5.4 Besonderheiten des NOS/VE-Vorrechners

5.4.1 Die CRAY-Kommando-Umgebung

Die Verbindung von der CD CYBER 930 zur CRAY wird auf der CYBER 930 über die von der Fa. CRAY bereitgestellte "NOS/VE Link Software" abgewickelt. Dem Benutzer stehen mit der CRAY-Kommando-Umgebung (CRAY Environment) eine Reihe von Kommandos zur Verfügung, um die einzelnen Transferleistungen anzufordern. In diese Umgebung gelangt man mit Hilfe des NOS/VE-Kommandos

ENTER_CRAY_ENVIRONMENT (ENTCE).

Das Kommando ENTCE bewirkt zunächst einen Suchvorgang und, wenn vorhanden, ein Abarbeiten der Datei:

$USER.CRAY_ENVIRONMENT_PROLOG.

In diese Prolog-Datei kann der Benutzer Kommandos ablegen, welche nach Aufruf von ENTCE ausgeführt werden, und er kann Variable definieren, die im späteren SUBMIT-Kommando ihre Verwendung finden.

Beispiel: Der Inhalt der Datei $USER.CRAY_ENVIRONMENT_PROLOG sei folgender:

```
DISV $TIME
USER_NUMBER='bfabcdef'
USER_PASSWORD='profi'
```

Bei Aufruf von ENTCE wird die Uhrzeit ausgegeben und die User-Variablen werden eingerichtet. Die soeben eingerichtete Variablenliste erhält man mit dem Kommando DISVL:

```
LOCAL VARIABLES IN UTILITY cray_environment:

USER_PASSWORD     USER_NUMBER
```

Der Prompting-String von ENTCE ist *ce/*. Die UNICOS-Kommando-Umgebung wird mit QUIT verlassen. Unter ENTCE sind am ZIB folgende Kommandos möglich:

DISPLAY_CRAY_STATUS	DISPLAY_JOB
DISPLAY_JOB_INFORMATION	DISPLAY_STATION
DISPLAY_STATION_INFORMATION	DISPLAY_STATION_MESSAGES
DISPLAY_STATION_PARAMETERS	DROP_JOB
KILL_JOB	QUIT
REPLY_TO_STATION_MESSAGE	RERUN_JOB
SELECT_JOB	SELECT_STATION
SET_JOB_SENSE_SWITCH	SUBMIT_CRAY_JOB
TEXT_FIELD	TEXT_FIELD_IN
TEXT_FIELD_PR	

Die wichtigsten Kommandos sind bereits in den Kapiteln 5.1 und 5.3 beschrieben.

Bemerkung: Die "Kommandos", die mit "TEXT_FIELD" anfangen, sind keine echten SCL-Kommandos. Sie dienen nur als Parameter für DISPLAY_COMMAND_INFORMATION (DISCI), um die Syntax der Textparameter in den Anweisungen *acquire*, *fetch* und *dispose* zu erhalten.

5.4.2 Besonderheiten des NOS/VE-Dateitransfers

Der Dateitransfer zwischen CYBER 930 und CRAY geschieht mit den UNICOS-Kommandos *dispose*, *acquire* und *fetch*. Im folgenden wird der für den CRAY-Vorrechner CYBER 930 spezielle Textparameter *-t* der drei Transferkommandos beschrieben. Über die für die eigene Benutzerkennung geltenden Default-Werte der Transfer-Parameter informiere man sich mittels des Kommandos

DISPLAY_JOB_ATTRIBUTES DO = ALL.

Textparameter für **-dST** (permanente Datei) bei ***acquire*** und ***fetch*** sowie ***dispose***; Auswahl:

FILE, F = file
PROLOG = string
EPILOG = string
LOGIN_USER, USER, U, LU = name
PASSWORD, PW = name
LOGIN_FAMILY, LF, FAMILY_NAME, FN = name
LOGIN_ACCOUNT, LA, ACCOUNT, A = name
DISPLAY_TRANSFER_JOB_LOG, DTJL = boolean
FILE_CONTENTS, FILE_CONTENT, FC = name oder keyword

Parameterbeschreibung:

FILE — NOS/VE-Dateiname im Transfervorgang (in NOS/VE-Schreibweise; keine Abbildung von Sonderzeichen).
Fehlt diese Angabe, wird der Wert des Parameters *-n* (Filename am Front-End) verwendet. Fehlt auch dieser, wird der *UNICOS local_path* verwendet. In diesem Fall werden bestimmte Sonderzeichen in NOS/VE-Dateinamen auf andere Sonderzeichen abgebildet:

/	wird zu .
.	wird zu _

(Siehe auch CDC NOS/VE Link Software Command Reference Manual SC-0270 B, Kapitel 2.9).

PROLOG — Zeichenfolge, die als NOS/VE-Kommandofolge vor dem Dateitransfer ausgeführt wird.

EPILOG — Zeichenfolge, die als NOS/VE-Kommandofolge nach dem Dateitransfer ausgeführt wird.

LOGIN_USER — NOS/VE Benutzerkennung (Default: eigene).

PASSWORD — NOS/VE-Password für LOGIN_USER (Default: eigenes).

LOGIN_FAMILY — NOS/VE-Family für LOGIN_USER (Default: eigene).

LOGIN_ACCOUNT — Account-Name für LOGIN_USER (Default: eigener).

DISPLAY_TRANSFER_JOB_LOG

TRUE	NOS/VE-Jobprotokoll wird nach *stderr* umkopiert.
FALSE	NOS/VE-Jobprotokoll wird unterdrückt (Default).

FILE_CONTENTS — Gibt den Datentyp eines File-Inhaltes an (vergl. NOS/VE File-Attribute). Es kann ein Name oder eines der folgenden Schlüsselwörter sein:

LEGIBLE	Datei im ASCII-Code mit ASCII-Drucksteuerung (Default).
LIST	Datei im ASCII-Code mit ASA-Drucksteuerung (siehe Kapitel 3.4).
OBJECT	Binäre Information.
UNKNOWN	Dateityp nicht spezifiziert.

Beispiele:

```
fetch lfn1 -t'F=LFN2'
fetch lfn1 -n lfn2
```

Der Katalog für Transferjobs ist $USER, daher wird durch beide Kommandos jeweils die Datei *$USER.LFN2* vom Vorrechner CYBER 930 zur CRAY gebracht und dort unter dem Namen *lfn1* bereitgestellt.

Achtung: für den *n*-Parameter sind nur 15 Zeichen zulässig!

Beispiel:

```
fetch lfn1 -n lfn2 -t'LU=FREMD PW=GEHEIM LA=ACC'
```

Der zu *lfn2* gehörige LOGIN_USER sei in diesem Beispiel ein anderer als derjenige, der SUBMIT_CRAY_JOB startete; dies trifft insbesondere zu, wenn der Job über die Remote Station (siehe Kapitel 5.1.2), über DFN-RJE (siehe Kapitel 5.1.3) oder von einem anderen Vorrechner gestartet wurde.

Textparameter für **-d PR/PT/PU** beim **dispose** (Dateitransfer zur Druckausgabe); Auswahl:

COMMENT_BANNER, CB = string
COPIES, C = integer
FILE_CONTENTS, FC = name oder keyword
FORMS_CODE, FOC = string oder keyword
ROUTING_BANNER, RB = string
STATION, S = name oder keyword
USER_FILE_NAME, UFN = name

Hinweis: Viele der Parameter haben gleiche Namen und Funktionen wie im NOS/VE-Kommando PRINT_FILE.

Parameterbeschreibung:

COMMENT_BANNER Character-String, der mit dem zu druckenden File gesandt wird. Bei fehlender Angabe wird der Filename aus dem *dispose*-Kommando verwendet.

COPIES Legt die Anzahl der Kopien fest (1... 10).

FILE_CONTENTS Gibt den Datentyp eines File-Inhaltes an (vergl. NOS/VE File-Attribute). Es kann ein Name oder eines der folgenden Schlüsselwörter sein:

LEGIBLE	Datei im ASCII-Code mit ASCII-Drucksteuerung (Default).
LIST	Datei im ASCII-Code mit ASA-Drucksteuerung.

FORMS_CODE String (1 ... 6) zur Bezeichnung eines Druckers (Default: es gilt die Vorbesetzung der Job-Attribute). Die jeweilige am CDCNET angeschlossene Rechenanlage wird durch den Parameter STATION spezifiziert.

ROUTING_BANNER Character-String, der mit der zu druckenden Datei geschickt wird. Default: USER_JOB_NAME (siehe SUBMIT_CRAY_JOB, Kapitel 5.1).

STATION Schlüsselworte ZIB_AUTOMATIC bzw. FUB_AUTOMATIC. Anwendung: ein am ZIB ablaufender Job schafft seinen Output zur FUB, wenn dieses Jobattribut auf FUB_AUTOMATIC gesetzt ist.

USER_FILE_NAME Spezifiziert einen Namen, der der zu druckenden Datei mitgegeben wird. Default: der Wert des Parameters *-n* aus dem *dispose*-Kommando.

Beispiel:

```
dispose lfn1 -n lfn2 -t'S=FUB_AUTOMATIC'
```

Dieses Kommando bewirkt das Drucken der Datei *lfn1* auf einem der zentralen Drucker der NOS/VE-Anlage der FU Berlin. Der zugehörige Job kann von jeder Anlage des CDCNET mittels SUBCJ abgesetzt worden sein.

Textparameter für **-d IN** beim **dispose** (Dateitransfer als Jobinput); Auswahl:

LOGIN_FAMILY, FAMILY_NAME, FN, LF = name
OUTPUT_DISPOSITION, ODI, STANDARD_OUTPUT, SO = file oder keyword
USER_JOB_NAME, UJN, JOB_NAME, JN = name

Hinweis: Viele der Parameter haben den gleichen Namen und Funktion wie im NOS/VE-Kommando SUBMIT_JOB.

Parameterbeschreibung:

LOGIN_FAMILY Die für LOGIN_USER gültige NOS/VE-Family. Default: die im SUBMIT_CRAY_JOB verwendete.
OUTPUT_DISPOSITION (siehe SUBMIT_CRAY_JOB, Kapitel 5.1).
USER_JOB_NAME (siehe SUBMIT_CRAY_JOB, Kapitel 5.1).

Beispiel:

Innerhalb eines CRAY-Jobs wird ein Job zum NOS/VE-Vorrechner geschickt, der im Inputfile mitgeführt wurde. Der Standard-Output wird im Katalog $USER.$WAIT_QUEUE abgelegt. Als Filename wird die Bezeichnung von USER_JOB_NAME (Default, im folgenden Beispiel UJN = BZBUSER) genommen:

```
cat > job << 'Z'
LOGIN BZBUSER PASSWORD ZIB02
ENTCE
DISCS MF=VE
QUIT
LOGOUT
Z
dispose job -d IN -t'ODI=WQ'
```

Bemerkung: Mit Hilfe dieses Transferkommandos lassen sich auch wieder Kettenjobs konstruieren!

5.5 Besonderheiten des NOS/BE-Vorrechners

Besonderheiten des Dateitransfers bei NOS/BE

Im Kapitel 5.2 wurden die UNICOS-Kommandos *fetch*, *acquire* und *dispose* zum Transfer von Dateien vom bzw. zum Vorrechner allgemein beschrieben. Die Besonderheiten für den NOS/BE-Vorrechner werden hier aufgeführt:

Textparameter:

Für jeden Dateitransfer wird auf der NOS/BE-Anlage ein vollständiger Batchjob gestartet; die Steuerkarten für diesen Job müssen vom Benutzer als Textparameter des Transferkommandos angegeben werden. Die Zeichenfolge wird in " eingeschlossen, um die Auswertung des Fortsetzungszeichens \ durch die Shell zu ermöglichen (siehe 7.3.3). Sonderzeichen innerhalb des Textes (z.B. $), die nicht ausgewertet werden sollen, müssen durch ein vorangestelltes \ markiert werden (siehe das folgende Beispiel zu *acquire*).

Der Parameter *-t* darf maximal 240 Zeichen lang sein, der zu bildende NOS/BE-Job darf maximal 8 Steuerkarten enthalten, jedes Kommando darf nur einmal und mit maximal 80 Zeichen vorkommen. Die Anweisung CTASK dient dabei zum Kopieren der Datei und muß an der Stelle in den Steuerkarten stehen, an der eine Kopierroutine stehen würde:

CTASK,ALL.

Parameter:

ALL	Auflisten des NOS/BE-Protokolls (Jobdayfile) in *stderr*. Keine Angabe: Auflisten des NOS/BE-Protokolls (Jobdayfile) nur im Fehlerfall.

Unter NOS/BE wird die Datei unter dem durch *-n* festgelegten Namen angesprochen; fehlt *-n*, dann unter dem Pfadnamen, d.h. Pfadname und NOS/BE-Dateiname müssen übereinstimmen! In dem zu generierenden Job dürfen nur folgende NOS/BE-Steuerkarten vorkommen:

ACCOUNT, ATTACH, CATALOG, CONVERT, CTASK, FAMILY, FCOPY, GET, LABEL, LFTRANS, LIMIT, MOUNT, PURGE, PUT, REPLACE, REQUEST, REWIND, RJE, RST, VSN, XDRAW und XMIT.

Die für den Dateitransfer notwendige NOS/BE-Jobkarte im Textfeld darf keine Angaben zu T, IO, CM oder ST enthalten; zulässig sind nur die Parameter PW und NT.

Um die Möglichkeit zu haben, Dateien von ASCII nach Display oder umgekehrt zu wandeln, ist im Textfeld der Kommandos *acquire*, *dispose* und *fetch* das Kommando FCOPY zulässig:

FCOPY, P = lfn1, N = lfn2, PC = pc, NC = nc,R.

Parameter:

lfn1	Originaldatei.
lfn2	Konvertierte Datei.
pc	Code, in dem die Originaldatei interpretiert wird (möglich: ASCII8 oder DIS).
nc	Code, in dem die konvertierte Datei erstellt wird (möglich: ASCII8 oder DIS).
R	für *lfn1* und *lfn2* wird nach dem Konvertieren ein *rewind* durchgeführt.

Beispiel zu *acquire*:

```
acquire vektor \
-t "A8888.ACCOUNT,FU2034BB,,pw-825.ATTACH,VEKTOR,\
ID=8888.CTASK."   #Anstoß zum Filetransfer
```

Das Kommando *acquire* prüft zunächst, ob in der aktuellen Verzeichnis eine Datei namens *vektor* vorliegt. Ist eine vorhanden, ist die Abarbeitung von *acquire* beendet. Ist sie nicht vorhanden, wird auf der CYBER 825 ein Job unter der NOS/BE-Auftragsnummer 8888 gestartet, der die Datei dort sucht (*ATTACH*), die Blockung und Codierung vom NOS/BE-ASCII-Format in das UNICOS-Format umwandelt, sie zur CRAY schickt und in der aktuellen Verzeichnis unter *vektor* ablegt.

Beispiel für das Abspeichern einer Datei auf Magnetband am Vorrechner:

```
dispose matrix -fTR -dMT -t"A7777,NT1,PW=pw-825.\
LABEL,MATRIX,R,L=CRAYMATRIX,D=DE,VSN=W09999,RING.CTASK,ALL."
```

Das Kommando *dispose* schickt an die CYBER 825 einen Job, der die Datei *matrix* (die auch unformatiert sein kann) transparent (d.h. ohne Codekonvertierung und ohne Umblockung) im UNICOS-Format auf ein Magnetband schreibt (CD-NOS/BE-Tape, 9-Spur, 6250 bpi). Das Programm CTASK führt die eigentliche Übertragung durch.

Beispiel zum Transport zwischen CRAY und NOS/BE-Anlage mit Konvertierung durch FCOPY:

Eine Datei, die auf der NOS/BE-Anlage im Display-Code vorliegt, soll zur CRAY transportiert werden:

```
fetch get_display_dat -nASCII -fUD\
-t'x4711,PW=GEHEIM.ATTACH,DISPLAY,DISPLAYDATEI,ID=4711,'\
'PW=GEHEIMER.FCOPY,P=DISPLAY,N=ASCII,PC=DIS,NC=ASCII8,R.'\
'CTASK,ALL.'
```

Eine Datei soll auf der NOS/BE-Anlage im DISPLAY-Code abgelegt werden:

```
dispose send_as_display -nASCII -fUD -dST\
-t'x4711,PW=GEHEIM.LIMIT,5000.REQUEST,DISPLAY,*PF.'\
'CTASK,ALL.REWIND,ASCII.FCOPY,P=ASCII,N=DISPLAY,PC=ASCII8,NC=DIS,R.'\
'CATALOG,DISPLAY,DISPLAYDATEI,ID=4711,RD=GEHEIMER,PW=GEHEIMER.'
```

Beispiel zum Transport zwischen CRAY und NOS/BE-Anlage mit *lftrans*:

```
# QSUB-r lftrans
# QSUB-lM 300Kw
# QSUB-lT 5
. .setpw
#-----------------------------
# Inhalt des Files .setpw
# (dieses File sollte sich im Homeverzeichnis befinden)
#-----------------------------
#! /bin/sh
# Aufruf mit: . .setpw
set +x
BE_JOBPW=xxxxxxxxxx; export BE_JOBPW
BE_USER=378; export BE_USER
set -x
#-----------------------------
```

```
# Ende des Files .setpw
#------------------------------
set -v
set +x
dispose .profile -n profile -d ST -m BE -t "A$BE_USER,PW=$BE_JOBPW.\
CTASK,ALL.LFTRANS,PROFILE,CRAYFAM,DC=IF,JPW=$BE_JOBPW,TID=C6,DO=T."
exit
```

5.6 Informationen über abgegebene Jobs an der CRAY (RST)

Ein regelmäßiger Überwachungslauf zum Starten der Ausführung von Jobs (siehe Kapitel 3.3.2) erzeugt auch eine Datei, in der Informationen über den Verbleib der CRAY-Jobs abgelegt sind.

Diese Datei enthält über CRAY-Jobs in der *Input-Queue* folgende Informationen: Jobname, verwendeter Vorrechner, Rechenzeitanforderung T, Hauptspeicheranforderung M, SDS-Anforderung, Jobklasse, NQS-Nummer, Priorität P, Ressourcen-Priorität RP, Alterungspriorität AP und Userid.

Bei in Ausführung befindlichen CRAY-Jobs erhält man: Jobname, verwendeter Vorrechner, Restrechenzeit, aktuelle Hauptspeichergröße, aktuelle SDS-Größe, Jobklasse, Index, aktueller Job-Status und Userid.

Mit Hilfe des Programms RST können auf verschiedenen Rechnern sowohl im Dialog als auch im Batch Informationen über den Verbleib der CRAY-Jobs gelesen werden. Darüber hinaus können auch Informationen von Jobs auf anderen Rechnern ausgegeben werden.

Das RST-Programm wird in den verschiedenen Betriebssystemen folgendermaßen aufgerufen:

NOS/BE:

```
RST,JID = jobid,INST = institution,DC = dispositioncode,MF = mainframe,
      DMF = destinationmainframe,SMF = sourcemainframe,TID = terminalid,
      US = usernumber,CLASS = class,L = file,KEEP,COUNT,HELP.
```

NOS/VE:

```
RST
      job_id, jid, job_name, jn: any = $optional
      institution, inst: any of key zib, tub, fub, wrb, zi2, keyend,
            any, anyend = $optional
      disposition_code, dc: name = $optional
      mainframe, mf: name = $optional
      source_mainframe, smf: name = $optional
      terminal_id, tid: any = $optional
      destination_mainframe, dmf: name = $optional
      usernumber, us: any = $optional
      class, cl: any = $optional
      count, co: boolean = NO
      list, l, o: file = $optional
      keep, k: boolean = NO
      help, h: boolean = NO
      status: (VAR, BY_NAME) status = $optional
```

UNIX:

```
rst  -jid jobid  -dc dispositioncode  -us usernumber  -cl class  -h
```

Hinweise:

* Für wichtige Leistungen von RST sind Kurzformen möglich (siehe die Beispiele am Ende dieses Kapitels).

* In der Regel gilt, daß durch die Angabe von Parametern Einschränkungen bei der Auswahl der auszugebenden Jobs formuliert werden, d.h. bei fehlenden Parametern entfallen die Einschränkungen.

* Bei allen Parametern, die Zeichenfolgen als Werte zulassen, sind Teilfolgen und das Zeichen "*" erlaubt, z.B. liefert

```
US = ab**ef
```

Informationen zu Benutzerkennungen, die mit *ab* beginnen, gefolgt von zwei beliebigen Zeichen, und der Zeichenfolge *ef*, eventuell gefolgt von weiteren beliebigen Zeichen.

Parameter (Auswahl):

Hinweis: Die Namen eines Parameters in den verschiedenen Betriebssystemen werden in der Reihenfolge NOS/BE / NOS/VE / UNIX angegeben; ein "-" bedeutet "nicht vorhanden".

JID / JID / -jid — *job identification:* es werden nur Jobs ausgegeben, die dem angegebenen Wert (Jobname bzw. Auftragsnummer) entsprechen.

DC / DC / -dc — *disposition code*: es werden nur Jobs mit dem angegebenen Dispositionscode ausgegeben:

= EX	nur die Ausführungs-Warteschlange
= IN	nur die Eingabe-Warteschlange
= PR	nur die Druckausgabe-Warteschlange

MF / MF / - — *mainframe:* es werden nur Jobs ausgegeben, die sich auf der angegebenen Rechenanlage befinden:

= CRY	nur Jobs auf der CRAY des ZIB (default, falls MF ohne Wert angegeben wird)
= TUB	nur Jobs auf der CDC 835 und CDC 860 der TUB
= WRB	nur Jobs auf der CDC 825 des ZIB
= ZI2	nur Jobs auf der CDC 930 des ZIB

TID / TID / - — *terminal identification:* es werden nur Jobs ausgegeben, die die angegebene *Terminal_ID* aufweisen. Diese Angabe hat nur Sinn bei Statusinformationen über die NOS/BE-Anlage des ZIB.

US / US / -us — *userid:* es werden nur Jobs des angegebenen Benutzers ausgegeben.

CLASS / CLASS / -cl — *class*: es werden nur Jobs der angegebenen Klasse ausgegeben (Achtung: die Klassenbezeichnungen sind auf allen Rechnern unterschiedlich!).

HELP / HELP / -h — Bei Angabe dieses Parameters wird ein HELP-Text ausgegeben. Auf Rechenanlagen mit dem NOS/BE Betriebssystem ist dieser Parameter ein Schlüsselwortparameter ohne Parameterwert. Auf Rechenanlagen mit NOS/VE Betriebssystem ist es kein Schlüsselwortparameter; der anzugebende Parameterwert ist vom Typ *boolean* (Wert = YES, NO, ON, OFF, TRUE oder FALSE).

Erweiterte Statusmeldungen der CRAY-Jobs im RST

Der *Scheduler* auf der CRAY setzt erweiterte Statusmeldungen ab. Ein Job, der in der Klasse *warte* steht, bekommt sporadisch eine kleine Zusatzinformation, wenn dieser Job noch nicht rechnet, obwohl er eigentlich bereits rechnen sollte. Der Eintrag in der Spalte *class* wird linksbündig überschrieben (ursprünglich *warte*):

`ex/arte`	von diesem Benutzer rechnet bereits ein Job
`fl/arte`	die maximale Feldlänge aller rechnenden Jobs ist erreicht
`ti/arte`	die maximale Rechenzeit aller rechnenden Jobs ist erreicht
`pa/arte`	es werden bereits zu viele Jobs von diesem Partner ausgeführt
`sb/arte`	BE-Station inaktiv
`sv/arte`	VE-Station inaktiv
`s1/arte`	SUN-Station *serv01* inaktiv
`s2/arte`	SUN-Station *serv02* inaktiv

Beispiele für Aufrufe von einer NOS/BE-Anlage:

`RST,MF,DC = IN`	komplette CRAY Input Queue, nach der Priorität P geordnet.
`RST,9999`	alle Einträge zur NOS/BE-Auftragsnummer 9999.
`RST,MF,L = OUT`	alle CRAY Einträge gehen in die Datei OUT.

Beispiele für Aufrufe von einer NOS/VE-Anlage:

`RST MF DC = IN`	komplette CRAY Input Queue, nach der Priorität P geordnet.
`RST 9999`	alle Einträge zur NOS/BE-Auftragsnummer 9999.
`RST MF L = OUT`	alle CRAY Einträge gehen in die Datei OUT.

Beispiele für Aufrufe von einer UNIX-Anlage:

`rst -jid jid -dc IN`	komplette CRAY Input Queue, nach der Priorität geordnet.
`rst -jid 9999`	alle Einträge zur NOS/BE-Auftragsnummer 9999.
`rst > out`	alle CRAY Einträge gehen in die Datei OUT.

5.7 USCP-Kommandos

USCP-Kommandos sind UNICOS-Kommandos, die Informationen über das Ziel von Druckausgaben und die Herkunft von Jobs übernehmen oder verändern, die über die Vorrechner zur CRAY geschickt werden.

5.7.1 Abschicken eines Jobs von der CRAY zur CRAY

Mit dem Kommando *uscpqsub* besteht die Möglichkeit, von einem Job auf der CRAY einen neuen Job an NQS über USCP zu schicken. Dies bedeutet, daß der neue Job die Herkunft und die Validierung vom abschickenden Job übernehmen kann.

```
uscpqsub -m mf -i tid  file
```

Parameter:

-m mf	*mainframe:* Bezeichnung des Vorrechners (zwei Großbuchstaben), zu dem die Ausgabe des Jobs geschickt werden soll. Voreinstellung für *mf* ist die Herkunft des abschickenden Jobs.
VE	CYBER 930 (NOS/VE)

BE	CYBER 825 (NOS/BE)
-i tid	*terminal identification:* Angabe (bis zu acht Zeichen) für den Vorrechner, wohin die Ausgabe des Jobs geschickt werden soll. Voreinstellung für *tid* ist die Herkunft des abschickenden Jobs.
file	Name der Datei, die den neuen Job enthält.

5.7.2 Verändern der Information über das Ziel der Ausgabe eines Jobs

Das Kommando *uscproute* verändert die Information über das Ziel der Ausgabe eines Jobs. Dieses Kommando wirkt sich auf alle nachfolgenden *dispose*-Kommandos mit dem *disposition code PR* und auf die Standard-Ausgabe des Jobs *(stdout)* aus.

```
uscproute -m mf -i tid -d dc
```

Parameter (die Voreinstellung ist unterstrichen):

-m mf	*mainframe:* Bezeichnung des Vorrechners (zwei Großbuchstaben), zu dem die Ausgabe des Jobs geschickt werden soll. Voreinstellung für *mf* ist die Herkunft des Jobs.
VE	CYBER 930 (NOS/VE)
BE	CYBER 825 (NOS/BE)
-i tid	*terminal identification:* Angabe (bis zu acht Zeichen) für den Vorrechner über das Ziel der Druckausgabe des Jobs. Voreinstellung für *tid* ist die Herkunft des Jobs.
-d dc	*disposition code:*
PR	Druckausgabe auf dem Vorrechner
SC	Vernichten von Druckausgaben.

6. Integration der CRAY in das TCP/IP-Netz

6.1 Übersicht

Neben dem im Kapitel 5 beschriebenen Zugang zur CRAY X-MP über die CD-Vorrechner des ZIB bestehen für Benutzer, die an einer Workstation oder an einem sonstigen lokalen Rechner arbeiten, der über Internet mit dem ZIB-Internet in Verbindung steht, weitergehende und zumeist bequemere Möglichkeiten des Zugangs zur CRAY.

Auf dem lokalen Rechner muß eine Netzsoftware für die Unterstützung von TCP/IP (Transmission Control Protocol / Internet Protocol) installiert sein. Diese Software unterstützt neben einer Reihe von Zusatzleistungen im wesentlichen einen synchronen (also von einem Menschen zeitgleich kontrollierten) Dateitransfer und das Login und die Dialogführung auf einem entfernten Rechner. Obwohl die meisten Benutzer auf eine solche Software im Zusammenhang mit einer Workstation unter dem Betriebssystem UNIX und der Nutzung eines Ethernet-Zugangs stoßen werden, muß festgehalten werden, daß TCP/IP unabhängig von einem speziellen Betriebssystem und einer speziellen Netz-Technologie ist. Dies wird z.B. beim Betrieb eines lokalen Rechners unter MS-DOS deutlich, wobei gewisse Beschränkungen in der Nutzung von TCP/IP in Kauf genommen werden müssen. Immerhin ist hierbei noch ohne weiteres eine Nutzung der aktiven, auf TCP/IP basierenden Dienste möglich.

Eine Nutzung der TCP/IP-Software erfordert natürlich auf dem lokalen Rechner das Vorhandensein entsprechender Hardware für den Netzzugang.

Unabhängig von der Möglichkeit des Netzzugangs ist die Frage zu klären, ob über diesen Zugang die CRAY des ZIB zu erreichen ist. Dies wird beispielsweise für Teilnehmer am WOTAN-Netz der TUB und für eine Reihe von lokalen Rechnern innerhalb der FUB sichergestellt.

Informationen über die Verknüpfung der lokalen Netze im Norddeutschen Wissenschaftsbereich finden Sie in Kapitel 1.4.3; allgemeine Informationen über die Internet-Dienste im Kapitel 1.4.4.

Die drei Grundanwendungen, die auf dem INTERNET unterstützt werden, sind

FTP — *File Transfer Protocol* erlaubt es, Dateien zwischen zwei Rechnern zu übertragen. Da es viele verschiedene Dateiformate gibt, können gewisse Umsetzungen automatisiert werden.

TELNET — erlaubt den interaktiven Zugang zu Rechnern mit Netzwerk. Häufig werden sogenannte *Terminal Server* verwendet. Dies sind Geräte mit mehreren Terminalschnittstellen und einem Netzanschluß. Der Benutzer kann dann auch von einfachen Terminals mit TELNET einen beliebigen Rechner am Netzwerk erreichen.

SMTP — *Simple Mail Transfer Protocol,* mit dem zwischen zwei Rechnern elektronische Post ausgetauscht wird. Im Gegensatz zu X.400 oder UUCP wird dabei immer eine direkte Verbindung zum empfangenden Rechner aufgebaut. Die Adressierung und das Format der übertragenen Post wird im RFC822 festgelegt.

Zusätzlich stellt das ZIB folgende Anwendung zur Verfügung:

ALRJE — Einbringen von Batch-Jobs in die CRAY und, falls gewünscht, automatisches Zurücksenden der Jobausgabe sowie eventueller weiterer Ausgabedateien mit Hilfe des FTP-Dienstes.

6.2 Der Dateitransferdienst FTP

Das Kommando *ftp* ist die Benutzerschnittstelle für den Dateitransferdienst FTP (File Transfer Protocol) des ARPANET. Dieses Kommando erlaubt dem Benutzer, eine Datei von oder zu einem entfernten Rechner zu transportieren. In der Regel wird der Benutzer FTP an "seinem" lokalen Rechner (Local Host) aufrufen, da dieser Rechner im allgemeinen über gute Zugangsmöglichkeiten und komfortable Editiermöglichkeiten verfügt. FTP wird dann einen Transfer zu einem entfernten Rechner (Remote Host) durchführen.

Es wird vorausgesetzt, daß sich die zu übertragende Datei, hier z.B. die Datei *crayjob,* im aktuellen Katalog des Benutzers auf dem lokalen Rechner befindet. Zur Übertragung muß zunächst interaktiv FTP aufgerufen werden, wobei als Parameter die Internet-Adresse der CRAY *130.73.128.2* oder des "UNIX Front End Relay" *ufer 130.73.108.21* angegeben wird.

Wurden vorher keine weiteren Maßnahmen getroffen, so fordert FTP nun die Validierung des Benutzers durch Angabe von Login-Name und Paßwort an. Falls die Angaben akzeptiert werden, kann nun nach Eingabe-Aufforderung der eigentliche File-Transfer erfolgen, wobei zunächst die Pfadangabe für die Quelle und dann die Pfadangabe für das Ziel das Transfers gemacht werden, also z.B. mit

```
put crayjob job
```

In der nun folgenden Wartezeit findet die Übertragung der Dateien statt. Nach einer weiteren Eingabe-Aufforderung kann FTP durch die Eingabe von *quit* verlassen werden.

Da die über die Systemvariable *$PERM* von der CRAY aus erreichbaren Dateien auf der UNIX-Workstation *serv01* auch von *ufer* aus erreichbar sind, empfiehlt sich der Transfer über *ufer*, um die CRAY von Verwaltungsaufgaben zu entlasten. Bei größeren Dateien (z.Zt. bei Dateien, die größer als 10 MB sind) muß der Transfer über *ufer* erfolgen. Eine Ausnahme bilden Transfers von Jobs im ALRJE-Dienst (siehe Kapitel 6.3); diese müssen z.Zt. noch zur CRAY übertragen werden.

6.2.1 Hilfsdateien für den FTP-Dienst

Die Benutzung der TCP/IP-Dienste wird im allgemeinen durch einige Hilfsdateien unterstützt, die der Benutzer wahlweise anlegen kann. Hierzu wird teilweise eine Abstimmung mit dem System-Administrator des lokalen Rechners erforderlich sein.

Die Datei /etc/hosts

Die Angabe der Internet-Adresse der CRAY beim *ftp*-Aufruf kann als umständlich empfunden werden. Stattdessen kann dort ein frei wählbarer Name angegeben werden, der - auch bei anderen TCP/IP-Aufrufen - stellvertretend für die Internet-Adresse eines entfernten Rechners steht. Das Textfile */etc/hosts,* welches vom System-Administrator gewartet wird, stellt in jeder seiner Zeilen eine Beziehung zwischen einer Internet-Adresse und einem Namen her. Z.B. kann für die Rechner des ZIB, unter Verwendung der wirklichen Internet-Adresse, folgende Zeilen eingetragen werden:

```
130.73.128.2      cray        bzw.
130.73.108.21     ufer
```

Der *ftp*-Aufruf kann nun wie folgt aussehen:

```
ftp cray          bzw.
ftp ufer
```

Die Datei *.netrc*

Bevor eine Datei vom FTP übertragen wird, erfolgt zunächst eine Zugangsprüfung des Benutzers im Rahmen einer Login-Prozedur. Dieses Verfahren kann durch Einrichten einer Datei mit dem Namen *.netrc* im HOME-Katalog des Benutzers automatisiert werden. Jede Zeile des Textfiles *.netrc* enthält die Login-Information für einen entfernten Rechner. Sie hat das Format

machine rhost login rhostuserid name password rhostpassword

also z.B.

```
machine cray    login bt101234      password cosOmic
```

Die beim Aufruf von *ftp cray* erforderliche Login-Information wird nun dieser Datei entnommen. Die Eintragung der Paßwort-Information ist optional. Fehlt sie, so wird das Paßwort nachgefragt und man spart nur die Angabe des Login-Namens. Für *rhost* muß der in */etc/hosts* vereinbarte Name angegeben werden; eine Internet-Adresse ist hier unzulässig. Die Verwendung von *.netrc* setzt also einen Eintrag in */etc/hosts* voraus.

Wichtig: Die Datei *.netrc* darf nur Lese- und Schreibberechtigung für den Eigentümer haben, sonst wird jeder *ftp*-Aufruf abgewiesen! Dies erreicht man durch den Befehl

chmod 600 .netrc

Bei Eintragung des CRAY-Paßwortes in *.netrc* sollten die Sicherheitsrisiken dieser Maßnahme bedacht werden, die trotz eingeschränkter Leseberechtigung bestehen können.

6.2.2 Die wichtigsten Unterkommandos von ftp

Die wichtigsten Unterkommandos von *ftp* werden im folgenden aufgeführt. Die komplette Liste erhält man z. B. mit dem UNICOS-Kommando *man ftp*.

ascii	Setzt den Übertragungsmodus auf codiert (default).
binary	Setzt den Übertragungsmodus auf binär.
cd remdir	Setzt den Arbeitskatalog auf dem entfernten Rechner auf *remdir*.
close	Beendet die aktuelle Verbindung zwischen beiden Rechnern.
delete remfile	Löscht auf dem entfernten Rechner die Datei *remfile*.
dir remdir locfile	Gibt eine Liste mit dem Inhalt des Arbeitskatalogs auf dem entfernten Rechner in die Datei *locfile* oder, wenn diese fehlt, auf den Bildschirm aus. Fehlt *remdir*, so wird der aktuelle Katalog ausgegeben.
get remfile locfile	Überträgt die Datei *remfile* auf dem entfernten Rechner zum lokalen Rechner in die Datei *locfile*; ist *locfile* nicht angegeben, so wird in die Datei auf dem lokalen Rechner mit dem Namen *remfile* kopiert.
help command	Gibt Informationen über das Kommando *command* oder die Liste aller *ftp*-Kommandos aus.

ls remdir locfile	Gibt eine kurze Liste mit dem Inhalt des Arbeitskatalogs auf dem entfernten Rechner in die Datei *locfile* oder, wenn diese fehlt, auf den Bildschirm aus. Fehlt *remdir*, so wird der aktuelle Katalog ausgegeben.
open host	Baut eine Verbindung zum angegebenen entfernten Rechner *host* auf.
put locfile remfile	Überträgt die Datei *locfile* auf dem lokalen Rechner zum entfernten Rechner in die Datei *remfile*; ist *remfile* nicht angegeben, so wird in die Datei auf dem lokalen Rechner mit dem Namen *locfile* kopiert.
pwd	Gibt den Namen des Arbeitskatalogs auf dem entfernten Rechner aus.
quit	Beendet die FTP-Verbindung und beendet FTP.
rmdir remdir	Löscht auf dem entfernten Rechner den Katalog *remdir*.
user userid password	Identifiziert auf dem entfernten Rechner den Benutzer. Auf der CRAY ist die Angabe des korrekten Paßwords nötig.

Beispiel: Ein Benutzer auf dem lokalen Rechner *ave3* überträgt eine Datei *daten* zum entfernten Rechner *cray*. Auf der CRAY wird die Angabe des Benutzers *username* und des Paßwortes erzwungen. Zum besseren Verständnis ist die Benutzereingabe *kursiv* gedruckt:

```
ave3% ftp
ftp> open cray
Connected to sn118.
220 sn118 FTP server (Version 5.2 Mon Mar 5 16:08:00 MEZ 1990) ready.
Name (cray: username):CR
331 Password required for username.
Password:
230 User username logged in.
ftp> pwd
257 "/home/username" is current directory.
ftp> put daten
200 PORT command successful.
150 Opening ASCII mode data connection for daten.
226 Transfer complete.
local: daten remote: daten
533556 bytes sent in 12 seconds (44 Kbytes/s)
ftp> quit
221 Goodbye.
ave3%
```

6.3 Der Jobtransferdienst ALRJE

6.3.1 Überblick

Im Zuge der zunehmenden Vernetzung von Rechnern tritt bei Benutzern einer entfernten Rechenanlage (Remote Host) der Bedarf auf, die Batch-Leistung eines solchen Rechners möglichst einfach von einem lokalen Rechner (Local Host) nutzen zu können. Dabei zeichnet sich der lokale Rechner im allgemeinen durch gute Zugangs- und komfortable Editiermöglichkeiten aus, während der entfernte Rechner wegen seines hohen Batch-Durchsatzes in Anspruch genommen wird.

Diesem Bedarf wird mit dem Konzept des RJE (Remote Job Entry) Rechnung getragen, welches inzwischen in den unterschiedlichsten Rechnerumgebungen implementiert worden ist. Die gemeinsame Leistung dieser Systeme besteht in der Möglichkeit, eine Datei, welche einen Job für den entfernten Rechner enthält, von einem lokalen Rechner zu diesem zu transportieren und, nach Ablauf des Jobs, dessen Ausgabe-Datei an den lokalen Rechner geliefert zu bekommen. Eine unterstützende Leistung kann z.B. darin bestehen, daß auf dem lokalen Rechner Status-Informationen über den entfernten Rechner bereitgestellt werden, die dem Benutzer eine bessere Kontrolle des entfernt ablaufenden Jobs ermöglichen. Auch kann es die Möglichkeit geben, vom lokalen Rechner aus den entfernten Job anzuhalten oder abzubrechen.

Für Benutzer, die z.B. von der NOS/BE-Anlage der TUB aus die CRAY-Anlage des ZIB benutzen wollen, steht seit geraumer Zeit der DFN-RJE-Dienst zur Verfügung, der im wesentlichen den oben beschriebenen Leistungsumfang umfaßt. In diesem Fall ist der lokale Rechner ein größerer Universalrechner, der von der Universität zentral betrieben wird. Im Zuge der Dezentralisierung von Rechenleistung und dem Aufkommen von lokalen Netzen (Local Area Networks, LANs) erhält der Benutzer zunehmend Zugang zu (kleineren) Rechnern, die sich in unmittelbarer Nähe seines Arbeitsplatzes befinden und deren Verfügbarkeit er direkt kontrollieren kann. Es entsteht der naheliegende Wunsch, nun einen solchen Rechner im Sinne von RJE zu seinem lokalen Rechner zu machen, ihn also zur Erstellung und Verwaltung von Jobs zu benutzen, die auf dem entfernten Rechner, in diesem Fall der CRAY, laufen sollen.

Eine Implementierung des RJE-Konzeptes setzt normalerweise das Vorhandensein spezieller Software sowohl auf dem entfernten als auch auf dem lokalen Rechner voraus, die beide beim Übertragen des Jobs und der Rücklieferung der Ausgabe-Datei geeignet zusammenarbeiten müssen. Diese Spezialisierung ermöglicht einerseits eine nahtlose Einbettung in die Systemumgebung, sie erfordert andererseits einen hohen Entwicklungsaufwand, der die Zahl der unterstützten Systeme und damit die Verfügbarkeit des Dienstes einschränkt.

Mit der Bereitstellung von ALRJE (Alternative RJE) bietet das ZIB einen RJE-Dienst an, der auf der Seite des lokalen Rechners nur das Vorhandensein der standardisierten, inzwischen weitverbreiteten TCP/IP-Protokolle vorsieht (siehe Kapitel 1 und 6.2). Es kann davon ausgegangen werden, daß praktisch alle im Bereich von FUB und TUB betriebenen Workstations diese Bedingung erfüllen oder aber - bei Bedarf - in naher Zukunft mit vergleichsweise geringem Aufwand erfüllen werden. Damit wird in kurzer Zeit eine breite Verfügbarkeit dieses Dienstes erreicht.

Das Konzept des ALRJE besteht darin, eine Datei, welche den CRAY-Job enthält, von dem lokalen Rechner in ein spezielles, nur dem jeweiligen Benutzer gehörenden Dateikatalog auf der CRAY zu übertragen. Dort wird es von einem Service-Programm aufgegriffen und in die CRAY-Eingabeschlange gestellt. Die Übertragung zur CRAY erfolgt mit dem Filetransfer-Dienst FTP, der im allgemeinen Teil einer TCP/IP-Implementierung ist. Die Validierung, also die Prüfung von Benutzerkennung und Paßwort, findet vor der Übertragung der Datei statt und ist eine Standardleistung derjenigen FTP-Komponente, die die Datei entgegennehmen soll.

Die Nutzung von ALRJE kann in zwei Stufen erfolgen: Die Minimalstufe umfaßt die Entgegennahme und Ausführung eines Jobs durch die CRAY und die Ablage der Ausgabe-Datei in einem Katalog auf der CRAY. Die zweite Stufe beinhaltet das automatische Versenden der Ausgabe-Datei an einen anderen Rechner, welcher im allgemeinen der lokale Herkunfts-Rechner des Jobs sein wird. Die Nutzung der zweiten Stufe erfordert weitergehende vorbereitende Maßnahmen. Der Anwender kann aus Gründen, auf die später eingegangen wird, auf die Nutzung der zweiten Stufe verzichten, ohne den grundsätzlichen Gewinn dieses Dienstes preisgeben zu müssen.

Um diese Beschreibung für den Einstieg nicht mit Einzelheiten zu überfrachten, wird zunächst auf die Minimalstufe eingegangen. Beim Übergang auf die zweite Stufe werden die dazu erforderlichen Voraussetzungen beschrieben.

6.3.2 Antrag auf Benutzung des ALRJE-Dienstes

Für die Nutzung des ALRJE-Dienstes benötigt der Anwender einen nur ihm zugänglichen ALRJE-Katalog, welcher zu diesem Zweck auf Antrag vom Anlagenbetrieb des ZIB eingerichtet wird. Der Antrag ist an Herrn Busch zu richten, telefonisch ((030) 89 604 135) oder mit elektronischer Post über folgende Adressen:

```
        RFC-822     : busch@sc.zib-berlin.dbp.de
oder    X.400       : C=de;A=dbp;P=ZIB-Berlin;OU=sc;S=Busch
oder    SMTP        : busch@ave
```

Entscheidend für die Namensgebung beim ALRJE-Katalog ist der *login name* des Benutzers auf der CRAY. Dabei handelt es sich um diejenige Kennung, die der Benutzer am Beginn eines Batch-Jobs in der Zeile

```
# USER = userid PW = password
```

macht, also z.B. *bt101234*. Insgesamt sind folgende Angaben erforderlich:

user name	-	Vor- und Zuname des Benutzers
phone	-	Telefonnummer
local host	-	Name des lokalen Rechners, wie er in einer Netzumgebung üblicherweise vergeben wird
local hosttype	-	Gerätetyp und Betriebssystem des lokalen Rechners
Internet address	-	Internet-Adresse des lokalen Rechners
mail address	-	Mail-Adresse des Benutzers auf dem lokalen Rechner

local host ist ein Name, der stellvertretend für die Internet-Adresse des lokalen Rechners steht und unter dem dieser Rechner der CRAY bekannt ist.

6.3.3 Der ALRJE-Dateikatalog

Nach Antragstellung wird für den Benutzer auf der CRAY ein ALRJE-Dateikatalog eingerichtet. Er hat den Pfadnamen */tmp/alrje/login name,* also z.B. */tmp/alrje/bt101234* und ist nur dem jeweiligen Benutzer zugänglich. Ein ALRJE-Job muß vom Benutzer in diesem Katalog abgelegt werden. Dort wird er von einem Service-Programm aufgegriffen und in die Eingabe-Warteschlange der CRAY eingereiht. Die Validierung des Jobs findet vor Entgegennahme der Datei auf der CRAY durch die entsprechende TCP/IP-Komponente statt. Befindet sich der Job erst einmal in dem ALRJE-Dateikatalog, findet keine weitere Prüfung statt. Daraus ergibt sich die Forderung nach einer hohen Aufmerksamkeit, die der Benutzer der Zugangssicherung seines ALRJE-Verzeichnisses widmen sollte. Ein fremder Benutzer, der Zugang zu diesem Katalog gewinnt, kann die entsprechende Benutzerkennung so benutzen, als ob ihm das Paßwort des Benutzers bekannt wäre.

Der Pfadname des ALRJE-Verzeichnisses befindet sich in einer UNICOS-Umgebungsvariablen mit dem Namen *$ALRJE.*

6.3.4 Struktur eines ALRJE-Jobs

Eine Datei, welche einen ALRJE-Job enthält, muß in der ersten oder zweiten Zeile eine QSUB-Anweisung erhalten. Diese Anweisung besteht aus dem UNIX-Kommentarzeichen (#) in Spalte 1, gefolgt von der Zeichenkette *QSUB* sowie einer zugehörigen Option. Die Optionen wirken für die Jobsteuerung im allgemeinen so wie bei einer Eingabe des Jobs über den DFN-RJE, jedoch sind zwei Angaben für die Bearbeitung durch ALRJE von Bedeutung:

- Der mit *# QSUB -r request-name* optional anzugebende Request-Name ist für die automatische Namensbildung der Ausgabe-Datei von Bedeutung. Darauf wird im Kapitel 6.3.6 näher eingegangen.

- Es sollte keine Umlenkung von *stderr* und *stdout* mit Hilfe der QSUB-Optionen *-e* und *-o* erfolgen. ALRJE kann dann diese Ausgabe-Dateien nicht in der beabsichtigten Weise bearbeiten.

Die bei Zugang über die auf den CYBER-Vorrechnern verfügbare Station-Software erforderliche Zeile

```
# USER = ... PW = ...
```

kann entfallen und wird bei Vorhandensein ignoriert. Ein Job, der den RJE-Zugang über die Vorrechner benutzt, kann im allgemeinen mit geringen Änderungen so gestaltet werden, daß er zusätzlich auch über ALRJE läuft.

Beispiel:

```
# USER=bt101234 PW=cosOmic    # wird ignoriert
# QSUB-r crayjob              # dient der Namensgebung für
# QSUB...                     # Ausgabe-Dateien
# QSUB...                     # weitere QSUB-Anweisungen ohne
# QSUB...                     #        die Optionen -e und -o
...
...                           # UNICOS-Script für den Job
...
```

6.3.5 Übertragung und Bearbeitung des ALRJE-Jobs

Die Übertragung der Job-Datei an die CRAY findet mit dem FTP-Dienst statt, so wie er in Kapitel 6.2 beschrieben ist. Als Zieldatei ist der ALRJE-Dateikatalog des Benutzers zu verwenden, also z.B.:

```
put crayjob /tmp/alrje/bt101234/crayjob
```

Das ALRJE-Service-Programm durchsucht in regelmäßigen Abständen die ALRJE-Kataloge aller Benutzer und reiht dabei aufgefundene Job-Dateien in die Eingabe-Warteschlange der CRAY ein. Zum Zeitpunkt der Erstellung dieser Beschreibung findet das Durchsuchen etwa alle fünf Minuten statt. Sobald sich der Job in der Warteschlange befindet, hat seine Herkunft über ALRJE im allgemeinen keinen Einfluß mehr auf seine Bearbeitung; er verhält sich dann z.B. so, als ob er über den DFN-RJE gekommen wäre.

Der Job läuft auf der CRAY unter der Benutzerkennung ab, die dem Namen des entsprechenden ALRJE-Dateikatalogs entspricht. Dies ist der Login-Name des CRAY-Benutzers. Nachdem der Job in die Warteschlange aufgenommen worden ist, wird die Job-Datei im ALRJE-Dateikatalog gelöscht.

6.3.6 Ausgabe-Dateien

Die von einem Job erzeugten Standard-Dateien *stderr* und *stdout* werden von ALRJE unter systematisch gebildeten Namen in dem zugehörigen ALRJE-Dateikatalog des Benutzers abgelegt. Die Namen werden aus dem *request-name* bzw. dessen ersten sieben Zeichen gebildet, wie er in einer QSUB-Anweisung angegeben werden kann (siehe Kapitel 6.3.4). Diese Zeichen bilden das Präfix, das heißt den ersten Teil der Datei-Namen.

Enthält der Job keine QSUB-Anweisung mit der Option -r, so wird das Präfix aus dem Datei-Namen des ALRJE-Jobs bzw. dessen ersten sieben Zeichen gebildet.

Die Namen der Standard-Dateien werden wie folgt gebildet:

stderr:	.e*nnnnn*	also z.B.	`crayjob.e12345`
stdout:	.o*nnnnn*	also z.B.	`crayjob.o12345`

nnnn ist der NQS-Request-Identifier des CRAY-Jobs. Er dient der eindeutigen Zuordnung von Ausgabe-Dateien zu einem bestimmten Job.

Der bis zu dieser Stelle dargelegte Leistungsumfang von ALRJE beschreibt die Minimalstufe der ALRJE-Anwendung (siehe Kapitel 6.3.1), die ohne kompliziertere Vorbereitungen vergleichsweise schnell ausgenutzt werden kann. Nach Ausführung des Jobs findet der Benutzer seine Ausgabe-Dateien in seinem ALRJE-Dateikatalog, wo er sie mit dem FTP-Dienst auf folgende Weise selbst abholen kann: Zunächst erfolgt wieder der Aufruf

```
ftp cray
```

Für die Verwendung eines Namens anstelle der Internet-Adresse und für das möglicherweise nachfolgende automatische Login gilt das in Kapitel 6.2 Gesagte. Nach erfolgreicher Validierung kann nach der Eingabe-Aufforderung z.B. folgende Zeile eingegeben werden:

```
get /tmp/alrje/bt101234/crayjob.e12345 crayjob.e45
```

Die Standard-Fehler-Datei des Jobs wird dann in den aktuellen Katalog des Benutzers mit dem an zweiter Stelle angegebenen Datei-Namen übertragen. Nach weiteren Eingabe-Aufforderungen können mit *get* weitere Dateien übertragen werden. Für eine Vielzahl weiterer *ftp*-Befehle, die die Arbeit unter Umständen sehr erleichtern können, wird auf Kapitel 6.2 oder das entsprechende Handbuch des jeweiligen Betriebssystems verwiesen. Nach Bearbeitung aller *ftp*-Anforderungen wird das Programm mit *quit* verlassen beendet.

Die Selbstabholung der Ausgabe-Dateien ist dann angezeigt, wenn der erhöhte Einrichtungs-Aufwand für das automatische Versenden vermieden werden soll oder wenn die Hard- oder Software-Eigenschaften des lokalen Rechners die automatische Zustellung erschweren oder unmöglich machen.

6.3.7 Automatisches Versenden der Ausgabe-Dateien

Das automatische Versenden der Ausgabe-Dateien setzt die Einrichtung von mindestens zwei Text-Dateien im HOME-Dateikatalog des Benutzers auf der CRAY und eine sorgfältige Abstimmung der Datei-Inhalte auf die jeweilige Netzumgebung voraus:

Die Datei *.netrc*

Diese Datei hat für die CRAY die entsprechende Funktion wie die in Kapitel 6.2.1 beschriebene für den lokalen Rechner. Die Datei ist für eine automatische Zustellung erforderlich, da das ALRJE-Service-Programm ein automatisches Login auf dem lokalen Rechner des Benutzers ausführen muß. Das Format der Datei wurde im gleichen Abschnitt dargestellt. Seine Einträge sind in gewissem Sinn invers zu den auf dem lokalen Rechner gemachten.

rhost ist jetzt der entfernte Rechner aus der Sicht der CRAY, also der lokale Rechner des Benutzers. Es muß sich um einen symbolischen Namen für den lokalen Rechner handeln. Dieser Name muß mit der Eintragung in der Datei */etc/hosts* auf der CRAY übereinstimmen. Die Eintragung wird vom CRAY-Betrieb auf der Grundlage der bei der Anmeldung zu *local host* gemachten Angabe vorgenommen.

login name ist eine Benutzerkennung, über die der Benutzer auf dem lokalen Rechner verfügt. Schließlich muß unter *password* das zugehörige Paßwort angegeben werden. Beispielsweise kann die Datei folgende Zeile enthalten:

```
machine meihost login meier password mypw
```

Wichtig: Die Datei *.netrc* darf nur Lese- und Schreibberechtigung für den Eigentümer haben, sonst wird jeder *ftp*-Aufruf abgewiesen! Dies erreicht man durch den Befehl

```
chmod 600 .netrc
```

Die Datei *.alhost*

In dieser Datei wird vereinbart, in welchen Katalog auf dem lokalen Rechner (aus Sicht der CRAY dem entfernten Rechner) die Ausgabe-Dateien des Jobs übertragen werden sollen. Die Datei darf nur eine Zeile enthalten, und zwar z.B.

```
machine meihost remdir /home/meier/alrje/
```

meihost ist der Name des lokalen Rechners, der bereits in den Dateien */etc/hosts* und *.netrc* aufgetreten ist. *remdir* ist eine Pfadangabe für den Katalog, in den die Ausgabe-Dateien übertragen werden sollen. Diese Angabe muß entsprechend der Syntax des Betriebssystems auf dem lokalen Rechner gemacht werden (siehe Kapitel 6.3.10). Bemerkenswert ist, daß die Angabe mit dem jeweils für Pfadangaben zulässigen Trennzeichen abgeschlossen werden muß. Vor Benutzung von ALRJE muß dieser Katalog auf dem lokalen Rechner eingerichtet werden.

Die beiden Dateien können auf dem lokalen Rechner erstellt und dann gemäß dem in Kapitel 6.2 beschriebenen Verfahren in den HOME-Dateikatalog auf der CRAY gebracht werden. Fehlen diese Dateien, so findet kein automatisches Versenden der Ausgabe-Dateien statt.

6.3.8 Automatisches Versenden an verschiedene Empfänger

Mit den bis hierher dargestellten Mitteln ist es nur mit hohem Aufwand möglich, Ausgabe-Dateien von verschiedenen Jobs an verschiedene Adressen zu verschicken. Dies kann jedoch einfach erreicht werden, indem der Benutzer in seinem ALRJE-Dateikatalog mehrere Dateien vom Typ der Datei *.alhost* (siehe Kapitel 6.3.7) einrichtet. Dabei kann jedem aus dem *request-name* gebildeten Präfix (siehe Kapitel 6.3.6) eine andere Empfangsadresse zugeordnet werden. Alle Ausgabe-Dateien mit dem gleichen Präfix haben also die gleiche Empfangsadresse. Soll die Ausgabe verschiedener Jobs zu verschiedenen Adressen geschickt werden, müssen die *request-names* der Jobs, falls angegeben, verschieden gewählt werden, oder die Job-Dateien müssen mit verschiedenen Datei-Namen übertragen werden.

Der Name für eine bestimmte Jobgruppe lokaler *.alhost*-Dateien wird mit dem Format *prefix.alhost* gebildet. Die Datei enthält die Zieladresse für alle Ausgabe-Dateien mit dem Präfix *prefix*. Das Format der Datei ist identisch mit der in Kapitel 6.3.7 beschriebenen Datei *.alhost*. Für jeden in einem *prefix.alhost* genannten entfernten Rechner (aus der Sicht der CRAY) muß sich genau ein Eintrag in der Datei *.netrc* im HOME-Dateikatalog des Benutzers auf der CRAY befinden. Der dort angegebene symbolische Name *rhost* muß in der CRAY-Datei */etc/hosts* (siehe Kapitel 6.3.7) eingetragen sein.

Eine einfache Art des Einrichtens der Datei *prefix.alhost* besteht darin, die Datei von einem entsprechenden Job aus zu schreiben. Dies kann z.B. auf folgende Weise geschehen:

```
# USER=bt101234 PW=cosOmic    # wird ignoriert
# QSUB-r newjob               # dient der Namensgebung für
# QSUB...                     # Ausgabe-Dateien
# QSUB...                     # weitere QSUB-Anweisungen ohne
# QSUB...                     #         die Optionen -e und -o
...
...                           # UNICOS-Script für den Job
...
echo "machine meihost2 remdir /home/meier/alrje/" \
     > /$ALRJE/$QSUB_REQNAME.alhost
...
```

Der Job richtet eine Datei *newjob.alhost* im ALRJE-Dateikatalog des Benutzers ein. Diese Datei wird für alle Jobs des Benutzers ausgewertet, für deren Ausgabe-Dateien das Präfix *newjob* gebildet wird. *.alhost*-Dateien werden nach erfolgreicher Übertragung der zugehörigen Datei-Gruppe gelöscht.

Ist die Information in einer Datei *prefix.alhost* fehlerhaft, so werden die zugehörigen Ausgabe-Dateien auch dann nicht übertragen, wenn eine gültiges Datei *.alhost* im HOME-Dateikatalog des Benutzers existiert. Es findet dann die übliche Behandlung wie bei Auftreten eines Übertragungsfehlers statt.

6.3.9 Prüfung des Datei-Transfers und Fehlerbehandlung

Um die Ausgabe-Dateien eines Jobs zu versenden, startet das ALRJE-Service-Programm einen oder mehrere FTP-Prozesse. Die Ausgabe-Dateien dieser Prozesse werden im ALRJE-Dateikatalog des Benutzers abgelegt, um sie bei der nächsten Sichtung des Verzeichnisses im Hinblick auf den Erfolg des Übertragungsversuches untersuchen zu können. Die Datei-Namen haben folgendes Format:

`ftp.o`*mmmmm* für *stdout*
`ftp.e`*mmmmm* für *stderr*

wobei *mmmmm* die NQS-Nummer des ALRJE-Service-Programms ist.

War die Übertragung erfolgreich, so werden alle übertragenen Dateien, die FTP-Ausgabe-Dateien und, falls vorhanden, die zugehörige *prefix.alhost*-Datei aus dem ALRJE-Verzeichnis gelöscht.

Wird bei der Untersuchung der FTP-Dateien festgestellt, daß bei der Übertragung ein Fehler aufgetreten ist, so wird diese Situation durch ein Umbenennen von Dateien gekennzeichnet, und zwar von

	`prefix.o`*nnnnn*	in	`prefix.O`*nnnnn*
bzw. von	`prefix.e`*nnnnn*	in	`prefix.E`*nnnnn*

Damit werden sie der normalen ALRJE-Behandlung entzogen. Die FTP-Ausgabe-Dateien werden umbenannt von

	`ftp.o`*mmmmm*	in	`ftp.O`*mmmmm*
bzw. von	`ftp.e`*mmmmm*	in	`ftp.E`*mmmmm*

Es wird dann versucht, die umbenannten FTP-Ausgabe-Dateien zum lokalen Rechner zu übertragen, und zwar zunächst in den in der Datei *.alhost* angegebenen Katalog, bei Mißerfolg in den HOME-Dateikatalog des Benutzers. Dem Benutzer wird auf der CRAY und auf seinem lokalen Rechner über Mail eine Nachricht über die Fehlersituation zugestellt, falls die Mail-Adresse dieses Rechners beim Beantragen des ALRJE-Dienstes (vergl. Kapitel 6.3.2) dem ZIB bekannt gegeben wurde.

Neben der Möglichkeit, durch elementare FTP-Aufrufe auf dem lokalen Rechner des Benutzers Ausgabe-Dateien auf diesen zu transferieren (wie in Kapiel 6.3.6 beschrieben), kann man auch durch Abschicken eines ALRJE-Dummy-Jobs ein neuerliches Versenden durch ALRJE initiieren. Dazu sollte sichergestellt sein, daß die generellen Übertragungsprobleme nunmehr behoben sind.

Dieser Dummy-Job bewirkt mit der Übertragung seiner eigenen Ausgabe-Dateien gleichzeitig den Transfer aller Dateien mit gleichen Präfix. Zu diesem Zweck muß er einen *request-name* haben, der zu dem gleichen Präfix führt wie dasjenige der noch ausstehenden Ausgabe-Dateien.

Sollen z.B. nicht übertragene Dateien mit dem Präfix *crayjob* abholt werden, kann dazu folgender Dummy-Job dienen:

```
# QSUB-r crayjob
date
```

6.3.10 Hinweise für Benutzer von lokalen Rechnern unter anderen Betriebssystemen

Die von ALRJE automatisch erzeugten Namen für die Ausgabe-Dateien eines Jobs enthalten zur Strukturierung einen Punkt. Dieses Zeichen ist in UNIX-Datei-Namen durchaus gebräuchlich und führt bei den verschiedensten Anwendungen kaum zu Problemen. Bei Übertragung der Dateien in andere Betriebssysteme kann sich der Punkt störend auswirken, z.B. wenn er dort als Element eines Datei-Namens oder einer Pfadangabe eine besondere Bedeutung hat. Allgemeingültige Lösungen zu solchen Problemen können nicht gegeben werden, jedoch soll im folgenden in diesem Zusammenhang auf zwei Betriebssysteme eingegangen werden, die für Benutzer des ALRJE-Dienstes von besonderer Bedeutung sind. Ausgangspunkt sei als Beispiel der Ausgabe-Datei-Name *crayjob.o12345*.

NOS/VE

Unter diesem Betriebssystem hat der Punkt die Funktion eines Trennzeichens zwischen den Komponenten einer Pfadangabe, ähnlich der Bedeutung des Zeichens "/" unter UNIX. Damit wird der vor dem Punkt stehende Teil des Datei-Namens als der Name eines Dateikatalogs angesehen, in welchem eine Datei mit einem Namen, der dem hinteren Teil entspricht, angelegt werden soll. Es wird also versucht, im Katalog *crayjob* eine Datei *o12345* anzulegen. Um dies zu ermöglichen, muß der Benutzer für jedes auftretende *prefix* (siehe Kapitel 6.3.6) auf dem NOS/VE-Rechner einen entsprechenden Katalog einrichten. Dabei ist zu berücksichtigen, daß mindestens noch für FTP-Ausgabe-Dateien mit dem Präfix *ftp* ein entsprechender Katalog eingerichtet werden muß.

Der Ort dieser Kataloge muß übereinstimmen mit der Angabe in der entsprechenden Datei *.alhost* bzw. *prefix.alhost*. Hat der Benutzer dort z.B. als *remdir* den Pfad *:fub.fu987ab6.alrje.* (abschließenden Punkt nicht vergessen!) angegeben, so muß er, dem obigen Beispiel folgend, unterhalb des Katalogs *alrje* die Kataloge *crayjob* und *ftp* einrichten. Dort müssen auch die Kataloge für weitere Präfixe, falls erforderlich, eingerichtet werden.

MS-DOS

Unter diesem Betriebssystem ist der Punkt nicht, wie bei NOS/VE, Trennzeichen innerhalb einer Pfadangabe. Er ist Teil des Datei-Namens wie unter UNIX, jedoch ist seine Stellung innerhalb des Namens stark eingeschränkt. In Datei-Namen unter MS-DOS dürfen dem Punkt höchstens noch drei Zeichen folgen, die als sogenannte Datei-Extension eine besondere Bedeutung haben. Die von ALRJE gebildeten Datei-Namen sind also im allgemeinen keine zulässigen MS-DOS-Namen.

Die erforderliche Anpassung der Datei-Namen muß die auf dem MS-DOS-Rechner laufende FTP-Software vornehmen. Dies kann auf verschiedene Weisen geschehen: Beispielsweise schneidet eine uns bekannte FTP-Implementierung den Teil des ALRJE-Datei-Namens hinter dem Punkt bis auf drei Zeichen ab. Andere Implementierungen können sich anders verhalten. Ein Bild vom Verhalten der jeweils vorliegenden Software sollte sich der Benutzer durch Probieren machen.

Das Trennzeichen zwischen den Teilen einer Pfadangabe unter MS-DOS ist "\". Bei der Angabe eines Pfades in der Datei *.alhost* bzw. *prefix.alhost* darf eine Laufwerksbezeichnung vorangestellt werden, also z.B. *d:\cray\alrje* (auf abschließenden Schrägstrich achten!). Sollen die Dateien in den Root-Dateikatalog übertragen werden, so ist nur die Laufwerksangabe (z.B. *c:*) ohne Schrägstrich zu machen.

FTP-Implementierungen unter MS-DOS können im allgemeinen keine Dateien entgegennehmen, während beliebige andere Aktivitäten auf dem Rechner laufen. Vielmehr setzt der Empfang von Dateien meistens das Umschalten in einen *Server-Mode* voraus, in dem oft nichts anderes getan werden kann, als das Eintreffen der Dateien abzuwarten. Ein Übertragungsversuch von ALRJE, der den MS-DOS-Rechner nicht im Server-Mode erreicht, mißlingt.

Die hier dargestellten Besonderheiten schränken die Brauchbarkeit eines MS-DOS-Rechners für die automatische Zustellung von Ausgabe-Dateien ein. Neben der Möglichkeit, ganz auf diese Zusatzleistung zu verzichten, kann der Benutzer in Erwägung ziehen, einen UNIX-Rechner aus seiner Umgebung, der für ihn leichter zugänglich ist als die CRAY, als ALRJE-Empfänger zu benutzen.

6.4 Dateitransfer zwischen CRAY und den NOS/BE-Rechnern in der TUB

Für den Dateitransfer zwischen den NOS/BE-Anlagen im ZIB und in der TUB steht schon seit längerer Zeit das NOS/BE-Kommando LFTRANS zur Verfügung, welches auch im Textfeld eines *dispose*-Kommandos von der CRAY aus verwendet werden kann. Mit LFTRANS und mit den DFN-Dateitransfer-Kommandos kann jedoch die leistungsfähige Datenleitung zwischen ZIB und TUB (2 Mb/s) nicht genutzt werden, da diese Dienste auf X.25 aufbauen und z.Zt. zwischen ZIB und TUB die X.25-Datenleitung nur mit einer Transferrate von 64 Kb/s bedient wird. Insbesondere für große Dateien (größer als 500 KByte, entspricht ca. 800 Prus) wurden von der ZRZ der TU gemeinsam mit dem ZIB spezielle Kommandos geschaffen, die auch für den Dateitransfer zwischen CRAY und den NOS/BE-Rechnern in der TUB diese leistungsfähige Leitung verwenden. Der Dateitransfer ist nur für Textdateien (ASCII- oder NOS/BE-DISPLAY codierte Dateien) realisiert.

Dieser Dateitransfer benötigt eine unter dem Betriebssystem NOS/VE der Rechenanlage Cyber 960 (TUX) der ZRZ realisierte Dateitransferschnittstelle. Das Betriebssystem NOS/VE ist auf Grund von Hardwareengpässen für Benutzer nicht direkt verfügbar, wird jedoch z.B. für diesen Dateitransfer eingesetzt. Diese Schnittstelle nutzt die Netzeinbindung des NOS/VE-Systems und dessen Verbindung zum NOS/BE-System, das gleichzeitig auf derselben Anlage läuft (dual state). Um den Dateitransfer zu nutzen, benötigt der Benutzer keine NOS/VE-Kenntnisse und keine Kennung auf dem NOS/VE-System. Voraussetzung ist lediglich je eine Kennung auf dem NOS/BE-System und der CRAY.

Zwischen CRAY bzw. *ufer* und dem NOS/VE-System in der TUB wird die Datei mittels *ftp* übertragen, auch ohne daß der Benutzer über Kenntnisse von FTP verfügen muß. Dieser Dienst benötigt jedoch die ZIB-UNIX-Workstation *serv01*, deshalb sollte im CRAY-Job in den NQS-Anweisungen

```
# QSUB -ZIB s1      bzw.
# QSUB -ZIB s1BE    stehen.
```

Dieser Dateitransfer zwischen CRAY und dem NOS/BE-System der TUB ist nur ein Sonderfall des allgemeinen Dateitransfers zwischen einem FTP-fähigen Rechner innerhalb des TU-Netzes WOTAN, welches mit dem ZIB-Netz über die Datenleitung (2 Mb/s) in Verbindung steht, und dem NOS/BE-System. Die zugehörigen Kommandos lauten *wo2be*, *BE2WO* und *WO2BE*, dabei ist "2" als "to" zu interpretieren und "wo" steht für einen Rechner im WOTAN-Netz der TU. Da die Zugriffe auf Rechner im WOTAN-Netz und im ZIB-Netz auf Grund der Verbindung identisch sind, sind die CRAY und der "UNIX-front-end-relay" *ufer* wie WOTAN-Rechner ansprechbar.

Folgende Kommandos stehen zur Verfügung:

wo2be (auf CRAY und *ufer*):

Der Benutzer arbeitet auf der CRAY oder auf *ufer* und möchte eine Datei zum NOS/BE-System der ZRZ übertragen.

BE2WO (auf dem NOS/BE-System):

Der Benutzer arbeitet auf dem NOS/BE-System der ZRZ und möchte eine lokale Datei zur CRAY oder zu *ufer* übertragen. Beide Kommandos werden im folgenden näher erläutert.

WO2BE (auf dem NOS/BE-System):

Der Benutzer arbeitet auf dem NOS/BE-System der ZRZ und möchte eine Datei von der CRAY oder von *ufer* zum NOS/BE-System holen. Dieses Kommando wird hier nicht erläutert, Sie finden eine Beschreibung auf dem NOS/BE-System der TUB unter DOC,SYSTEM,WO2BE.

Versenden von großen Dateien von CRAY oder *ufer* zum NOS/BE-System der TUB:

wo2be unixfile parameter

Die zu übertragende Datei kann im NOS/BE-System als permanente Datei, als indirekte Datei in eine Family, in eine Punch-Queue, in eine Print-Queue oder in eine Input-Queue abgelegt werden. Parameter können durch Angabe von *-Schlüsselwort wert* verwendet werden. Das Leerzeichen hinter *-Schlüsselwort* ist notwendig. (z.B. *-jpw geheim*)

Für alle Parameter (bis auf *jpw*) können globale Voreinstellungen in der Defaultdatei (siehe Parameter *df*) erfolgen. Die Verwendung der auf der CRAY üblichen Environment-Variablen BE_JOBPW, BE_USER, BE_FAMILY, BE_TK, BE_XR und BE_RD (definiert für Aufrufe von *getbe* und *putbe*) wird unterstützt. Die Variable BE_FAMILY wird für den Parameter *pfn* verwendet. Angegebene Parameter haben jedoch immer Vorrang. Environment-Variablen werden vor Parametern aus der Defaultdatei berücksichtigt. Die Parameter werden zum Teil nur knapp beschrieben; weitere Einzelheiten findet man in der Globalbeschreibung zu *wo2be* (DOC, CRUTIL, WO2BE).

Parameter:

unixfile	Dateiname der zu übertragenden Datei (kein Schlüsselwort). Die zusätzliche Angabe eines Pfadnamens ist zulässig (z.B. *nosbe/daten*).
-dc	Dispositioncode auf der NOS/BE-Anlage für die Transferdatei.
if	Abspeicherung unter der durch *-pfn* spezifizierten Family.
pf	Abspeicherung als Permfile im PUBLIC-Set unter dem durch *-pfn* spezifizierten Namen.
pr	Einbringen der Transferdatei mit dem Kommando PRINT in der Output-Queue unter der durch *-dctid* spezifizierten Terminal-ID.

pu	Einbringen der Transferdatei in der Punch-Queue unter der durch *dctid* spezifizierten Terminal-ID.
in	Einbringen der Transferdatei in die Input-Queue unter der durch *dctid* spezifizierten Terminal-ID.
-pfn	Family oder Permfilename für die Abspeicherung der Transferdatei auf der NOS/BE-Anlage (max. 40 Zeichen).
UNIXFAM	Keine Angabe: *-pfn UNIXFAM* für *-dc if*, sonst *-pfn unixfile* (der Name wird ohne
unixfile	Pfadinformation den entsprechenden NOS/BE-Konventionen angepaßt).
-ifn	Name der mit REPLACE zu erzeugenden indirekten Datei für *-dc if*. Keine Angabe:
unixfile	*-ifn unixfile* (der Name wird ohne Pfadinformation den entsprechenden NOS/BE-Konventionen angepaßt).
-jpw	NOS/BE Jobpaßwort (obligat). Keine Angabe : Environment-Variable *BE_JOBPW*.
-tid	Terminal-ID. Dieser Terminal- bzw. User-Identifier des Benutzers ist immer anzugeben oder in der Defaultdatei (Parameter *df*) vorzubesetzen. Der Benutzer erhält unter der Terminal-ID auf der NOS/BE-Anlage eine Meldung über den erfolgten Transfer oder ein Fehlerprotokoll. Notfalls kann *-tid c* angegeben werden, falls die Terminal-ID nicht bekannt ist; dann wird eine Meldung anhand der Auftragsnummer abgesetzt, und ein Fehleroutput wird ggf. am zentralen Drucker in der TUB ausgegeben. Der Parameter dient außerdem als Default für den Parameter *-dctid*.
-jid	Auftragsnummer des Benutzers für den NOS/BE-Job (obligat).
-code	Code für die Transferdatei unter NOS/BE.
ascii	Groß-/Kleinschreibung; Default für -dc if/pf/pr/pu.
display	Großschreibung; Default für -dc in.
-id	Benutzeridentifikation, unter der die Transferdatei abgelegt werden soll (nur für *-dc if/pf*). Keine Angabe: Wert von *-jid*.
-prp	Parameter für das Kommando PRINT unter NOS/BE; 1 bis 64, evtl. bis 71 Zeichen (nur für *-dc pr*). Keine Angabe: keine zusätzlichen PRINT-Parameter. Für den Fall *-dc pr* können hier weitere Parameter des PRINT-Kommandos angegeben werden, mit dem die Datei in die Ausgabewarteschlange gestellt wird.
-dctid	Terminal bzw. User-ID (nur für *-dc pr/pu/in*). Keine Angabe: *Wert von -tid*.
-xr	Paßwörter für die Family oder die permanente Datei nach NOS/BE-Konventionen.
-tk	Keine Angabe: Es werden leere Paßwörter übergeben (Wirkung wie keine
-rd	Paßwortangabe).
-mtt	Meldung an den NOS/BE-Benutzer.
yes	
-df	Datei zum Setzen von globalen Defaults.
wo2be.def	

Es wird versucht, nicht angegebene und weitere Werte für Parameter aus der Defaultdatei zu lesen. Die Direktivendatei kann für alle Parameter (bis auf *jpw*) zeilenweise globale Voreinstellungen enthalten. Leerzeichen werden ignoriert. Kommentarzeilen sind durch das Zeichen # in der ersten Spalte einzuleiten.

Beispiel einer Defaultdatei:

```
# Das ist eine Defaultdatei für wo2be
-jid 4711
-ifn indirektes_file_von_der_cray
-tid c6
-pfn crayfamily
```

Beispiel:

Es soll eine Datei mit dem Namen GMFILE aus dem Home-Katalog des Benutzers *bt404711* von der CRAY in die Family GRAFIK,ID = 4711 zur NOS/BE-Anlage der TUB übertragen werden und dort unter dem indirekten Dateinamen CRAY_GMFILE im DISPLAY-Code abgelegt werden.

```
wo2be gmfile -jpw nosbepw -jid 4711 -tid terminalid -pfn grafik \
       -ifn cray_gmfile -code display
```

BE2WO auf dem NOS/BE-System der ZRZ

Transfer einer lokalen Datei von der NOS/BE-Anlage der TUB in ein angegebenes Verzeichnis auf der CRAY oder *ufer*.

Aufruf:

BE2WO,LFN = lfn,DF = df,RFN = rfn

Beschreibung der Parameterwerte:

lfn	lokaler Name der Transferdatei
df	Name der Direktivendatei. Keine Angabe: die Direktivendatei wird aus der TUBE-Datei ZZZDIR entnommen.
rfn	Name für die zu transferierende Datei am Zielrechner. Keine Angabe zu RFN und keine Angabe für REMOTEFN. in der Direktivendatei: die Transferdatei erhält am Zielrechner den beim Parameter LFN angegebenen Namen lfn (in Kleinbuchstaben). Bei Verwendung des Parameters RFN wird die Angabe für REMOTEFN. in der Direktivendatei ignoriert. Der Name der Transferdatei am Zielrechner besteht dann aus Großbuchstaben.

Der Aufruf von BE2WO geschieht zweistufig. Beim Erstaufruf ist kein Parameter anzugeben, es wird im TUBE-Editor eine vorbereitete Direktivendatei angeboten. Die Verwendung einer Direktivendatei ist notwendig, da auf der NOS/BE-Anlage keine andere Möglichkeit besteht, Informationen im ASCII95-Zeichensatz (Groß/Kleinschreibung) auf einfache Weise zu erfassen. Sie enthält Informationen, die zur Speicherung der Transferdatei auf der Zielanlage notwendig sind.

Beim Erstaufruf von BE2WO erhalten Sie folgende Bildschirmausgabe:

```
BITTE DAS DIREKTIVENFILE AUSFUELLEN
 UND
BE2WO,<LFN> AUFRUFEN

10=REMOTEDIR. =''
20=REMOTEFN. =''
30=USER ='LOGIN-KENNUNG'
40=PASSWORD ='******'
50=INTERNETADR. ='130.73.128.2'

EDIT FILE EXISTS ..
```

Die Direktivendatei wird mit TUBE im *f(ormat) as95* bearbeitet. Es enthält fünf Zeilen und wird beim interaktiven Aufruf von BE2WO syntaktisch vorgegeben. Zwischen den Apostrophen sind die entsprechenden Angaben einzufügen. Der Benutzer muß mindestens die obligaten Informationen der Zeilen 30 und 40 'LOGIN-KENNUNG' und '******' (Paßwort) ersetzen.

Beschreibung der Zeilen der Direktivendatei:

Zeile 10:
enthält den Namen des Remote-Verzeichnisses, d.h. die Angabe, unter welchem Pfad die Transferdatei abgespeichert werden soll (z.B: *cyberfiles/jobs*). Default: " (leer) - Homekatalog.

Zeile 20:
enthält den Namen der entfernten Datei, d.h. die Angabe, unter welchem Namen die Transferdatei abgespeichert werden soll. Default: *lfn*, der angegebene lokale Dateiname (in Kleinbuchstaben).

Zeile 30:
enthält die Login-Kennung des Benutzers auf dem Zielrechner (z.B. die Benutzerkennung *bt404711* auf der CRAY)

Zeile 40:
enthält das Paßwort des Benutzers auf dem Zielrechner.

Zeile 50:
enthält die Internet-Adresse des Zielrechners.. Default: CRAY mit der Internet-Adresse 130.73.128.2.

Nach dem Ausfüllen der Zeilen der Direktivendatei (vergessen Sie nicht, die veränderten Zeilen abzuschicken!) ist BE2WO erneut aufzurufen; diesmal mit der Angabe der lokalen Datei *lfn*. Das Eingeben des TUBE-Kommandos *s(ave),df,n,o* ist nicht notwendig; die Direktivendatei wird automatisch aus der TUBE-Datei ZZZDIR entnommen. Sie erhalten keine weitere Meldung über den Dateitransfer und können in der Regel mit einem erfolgreichen Transfer rechnen, wenn Ihre Angaben in der Direktivendatei korrekt sind und der FTP-Dienst auf dem Zielrechner empfangsbereit ist. Nach dem Ausfüllen der Direktivendatei kann man sich diese Datei unter einem selbstgewählten Namen mit *s,filename,n,o* für spätere BE2WO/WO2BE-Aufrufe aufheben (z.B. in einer Family abspeichern); insbesondere ist dies für Batch-Aufrufe von BE2WO/WO2BE notwendig (siehe Parameter DF).

Erfolgt innerhalb einer interaktiven Sitzung ein weiterer Aufruf von BE2WO mit einem anderen lokalen Dateinamen *lfn*, werden natürlich alle Angaben der Direktivendatei, insbesondere auch der Dateiname für den Zielrechner, erneut unverändert übernommen; d.h. die zuletzt abgeschickte Datei wird auf dem Zielrechner überschrieben, falls in Zeile 20 der Direktivendatei ein Name für die entfernte Datei angegeben wurde und der Parameter RFN nicht verwendet wurde. Bei Folgeaufrufen von BE2WO/WO2BE ohne Parameter bekommt man die letzte Direktivendatei (in der TUBE-Datei ZZZDIR) vom Editor für Änderungen angeboten, die man sich mit dem Kommando *l(ist),a* auflisten und anschließend bearbeiten kann.

Beispiel:

Es soll eine lokale Datei mit dem Namen FTNPR in den Home-Katalog des Benutzers *bt404711* auf die CRAY unter dem Namen *ftnprog.f* transferiert werden.

Beim Erstaufruf von BE2WO wird die Direktivendatei wie folgt ausgefüllt:

```
10=REMOTEDIR. =''
20=REMOTEFN. ='ftnprog.f'
30=USER ='bt404711'
40=PASSWORD ='craypw'
50=INTERNETADR. ='130.73.128.2'
```

dann wird BE2WO erneut aufgerufen:

```
BE2WO,FTNPR
```

Für die automatische Übertragung einer größeren Anzahl von indirekten Dateien aus einer Family steht das Kommando FBE2WO zur Verfügung (siehe DOC,FAMLIB,FBE2WO auf dem NOS/BE-System in der TUB).

6.5 Nachrichtendienste MAIL und SMTP

6.5.1 Der lokale Nachrichtendienst MAIL

An der CRAY X-MP des ZIB steht der Nachrichtendienst *MAIL* für das Hinterlegen von Nachrichten an andere CRAY-Benutzer und das Lesen und Löschen von erhaltenen Nachrichten zur Verfügung. Das ZIB benutzt *MAIL* zum Beispiel zur Benachrichtigung von Benutzern bei automatisch erfolgenden Eingriffen wie etwa dem Löschen überzähliger Jobs in der Input-Queue oder bei abgebrochenen Übertragungen beim ALRJE-Dienst.

Lesen von Nachrichten

Zum Lesen von Nachrichten steht das Kommando

```
mail -p
```

zur Verfügung; es kopiert alle vorhandenen Nachrichten für den Benutzer, der *mail* aufruft, in die Standardausgabe. Dabei verbleiben jedoch alle Nachrichten im Briefkasten des Benutzers. Da die Briefkasten-Datei */usr/mail/username* eines Empfängers mit dem Benutzerkennzeichen *username* diesem Benutzer gehört, kann man sich den gesamten Inhalt des Briefkastens auch mit *ftp* oder anderen Dateimanipulations-kommandos direkt umkopieren und ansehen. Es ist jedoch nicht möglich, den Inhalt der Briefkasten-Datei direkt zu löschen.

Wenn ein Benutzer eine Nachricht über *mail* erhalten hat, wird diese Nachricht vom ZIB-*profile* in die Standard-Ausgabe kopiert. Um im Batch-Betrieb bis zu zwei Nachrichten auf der Standardausgabe wiederzugeben und danach zu löschen, steht das Kommando

```
delmail
```

zur Verfügung. Liegen mehr als zwei Nachrichten im Briefkasten des Benutzers, so gibt diese Kommandofolge nur die beiden ältesten Nachrichten aus und löscht diese.

Hat der Benutzer keine Nachricht erhalten, so erhält er den Hinweis

No mail.

Ist der Benutzer sicher, jede Ausgabe eines CRAY-Jobs zu lesen, so kann er das Kommando *delmail* auch standardmäßig in Jobs oder den Benutzerprolog einbauen.

Verschicken einer Nachricht

Das Verschicken einer Nachricht an einen Benutzer mit dem Benutzerkennzeichen *username* auf der CRAY kann im Batch-Betrieb dadurch vorgenommen werden, daß man die Nachricht zunächst in einer Datei *nachricht* hinterlegt und dann die Standardeingabe des *rmail*-Kommandos aus dieser Datei *nachricht* umleitet:

rmail username < nachricht

Das Benutzerkennzeichen des Absenders und das Datum des Abschickens werden (zusammen mit anderen Informationen) der Nachricht von dem Kommando *rmail* in einem Kopf hinzugefügt, so daß sie nicht Bestandteil der Nachricht zu sein brauchen.

6.5.2 Das Versenden von Nachrichten an andere Rechner

Das Lesen und Verarbeiten von Nachrichten ist einem Rechner wie der CRAY, der nur im Batch-Betrieb zugänglich ist, nicht angemessen, da der Benutzer erst nach Erhalt des nächsten Batch-Jobs über eine evt. vorliegende Nachricht informiert wird. Diese Aufgabe gehört wie auch z.B. das Editieren von Dateien auf einen möglichst nahe beim Benutzer stehenden Rechner mit möglichst komfortabler Oberfläche. Zur Realisierung dieses Konzepts bezüglich der Nachrichten besteht beim Verschicken von Nachrichten mit *rmail* auf der CRAY die Möglichkeit, diese dem Benutzer an einem anderen Rechner zuzustellen, sofern dieser Rechner mit der CRAY geeignet verknüpft ist.

Es ist nicht vorgesehen, auf der CRAY Nachrichten von einem anderen Rechner her zu empfangen (was bei einem Batch-Rechner ja auch nicht sinnvoll ist).

SMTP-Empfängeradresse

Die einfachste Möglichkeit, Nachrichten von der CRAY an einen Benutzer an einem anderen Rechner zu schicken, besteht dann, wenn dieser Rechner über SMTP (simple mail transfer protocol, einer Anwendung der TCP/IP-Protokolle) verfügt und mit der CRAY über Internet verbunden ist (vgl. Kap. 1.4.3). Insbesondere für alle UNIX-Rechner im WOTAN-Netz der TUB und für die meisten UNIX-Rechner der FUB ist dies gewährleistet.

Zum Verschicken der Nachricht wird wiederum das Kommando *rmail* verwendet, jedoch wird die Empfängerangabe ergänzt um den Namen des Rechners, an dem die Benutzerkennung des Empfängers bekannt und eingetragen ist. Der tatsächliche Rechnername muß auf der CRAY bekannt sein (d.h. vom Betrieb in der Datei */etc/hosts* eingetragen sein).

rmail login-name@remote-host

login-name	Eingetragene Benutzerkennung des Empfängers auf dem entfernten Rechner
remote-host	Tatsächlicher Name des entfernten Rechners

Beispiel:

```
rmail karl@suntul7
Der Job test1711 ist fertig geworden
.
```

(die Angabe des Punktes in Spalte 1 beendet die Eingabe für *rmail*).

Da jeder Benutzer der CRAY auch auf dem UNIX front end relay *ufer* im Dialog arbeiten darf, kann auch dort MAIL von der CRAY empfangen werden. Aufruf zum Abschicken:

```
rmail $LOGNAME@ufer
Text der mail
```

X.400-Empfängeradresse

Da das ZIB über ein *Gateway* zum Umwandeln von Nachrichten vom SMTP-Format in das dem internationalen Standard entsprechende X.400-Format verfügt, kann auch ein Empfänger mit einer X.400-Adresse erreicht werden. Das folgende Beispiel möge dies verdeutlichen:

Der Empfänger sei über folgende X.400-Adresse erreichbar:

```
C=de;A=dbp;P=FU-Berlin;OU=ZEDAT;OU=FUB02;S=Meyer
```

dann muß die Empfänger-Angabe bei *rmail* wie folgt lauten:

```
rmail /s=Meyer/ou=FUB02/ou=ZEDAT/prmd=FU-Berlin/admd=dbp/c=de@x400-gate
```

Die X.400-Adresse des Empfängers wird in der für das ZIB-Gateway festgelegten Schreibweise (mit "/" und den Schlüsselwörtern *s*, *ou*, *prmd*, *admd*, *c*) anstelle des *login-name* und die Bezeichnung *x400-gate* als Name des Rechners eingetragen.

Nachsenden von Nachrichten (*.forward*)

Nachrichten können bequemer auf einem Dialogrechner als auf einem Batchrechner wie die CRAY verarbeitet werden. Die SMTP-Dienste sehen daher vor, daß man sich Nachrichten automatisch zu einem anderen Rechner "nachsenden" läßt. Man achte jedoch darauf, daß die neue Adreßangabe korrekt ist und Nachrichten an diese Adresse auch ankommen, da sonst die nachzusendende Nachricht an den Absender zurückgeht oder sogar gelöscht wird ("nicht zustellbar"). Der Wunsch, Nachrichten nachzusenden, wird mit Hilfe einer Datei *.forward* im Heimatkatalog realisiert, der Inhalt von *.forward* wird als neue Adresse gedeutet und es wird die Nachricht an diese Adresse weitergeleitet.

Beispiel für den Inhalt von *.forward*:

```
karl@suntul7
```

6.6 Der Dialogdienst TELNET

Auf Rechnern mit der TCP/IP-Software sowie auf Terminals an Internet-Terminalservern steht das Kommando *telnet* zum Aufbau einer Dialogverbindung zu einem entfernten Rechner zur Verfügung. Wenn auch die CRAY selbst nicht als Dialog-Rechner angeboten wird, so kann der Benutzer doch den UNIX front end relay *ufer* im Dialog erreichen, etwa um Dateien im Katalog *$PERM* anzusehen oder zu verändern.

Der einfachste Aufruf lautet:

```
telnet host
```

Beispiel:

```
telnet ufer
```

oder

```
telnet 130.73.108.21
```

(Die Bedeutung der Rechnernamen bzw. der Internet-Adressen findet man in Kap. 1.4)

Nach Beenden der Dialogsitzung wird die Verbindung automatisch abgebaut. Eine bestehende Verbindung kann auch mit einer speziellen Zeichenfolge, wenn nicht anders vereinbart mit ^] (control - eckige Klammer zu), abgebrochen werden.

Beispiel einer Dialogsitzung:

```
telnet ufer
Trying 130.73.108.21 ...
Connected to ufer.
Escape character is '^]'.
SunOS UNIX (ufer)
login: username
Password:
Last login: Thu Apr 12 14:49:27 from ave3
SunOS Release 4.0.3_EXPORT (CLIENTEL_3.60) #1: Fri Mar 2 16:31:09 MET
1990
You have mail.
Enter terminal type :
sun
ufer$ env
HOME=/bz/username
LOGNAME=username
PATH=:/usr/ucb:/bin:/usr/bin
PERM=/bz/username
SHELL=/bin/sh
TERM=sun
USER=username
ufer$ ls -alg
total 11
drwxr-x--- 2 username username  512 Apr 12 14:41 .
drwxrwxr-x109 ndisk storadmi  2560 Apr 4 14:04 ..
-rwxr-xr-x 1 username username  4819 Apr 4 13:51 daten
-rwxr-xr-x 1 username username  1088 Apr 12 14:52 fort.1
ufer$ exit
Connection closed by foreign host.
```

7. Prozeduren in der Bourne-Shell

Der Text dieses Kapitels wurde mit geringfügigen Änderungen der Schrift "Einführung in UNIX" von H. Alt und M. Mitchelmore (LRZ München) entnommen.

7.1 Prozeduren

7.1.1 Die UNIX-Shells

Es gibt in UNICOS, wie in anderen UNIX-Systemen auch, mehrere Arten von Shells, z.B. die *Bourne-Shell* und die *C-Shell* (siehe Kap. 4). Die voreingestellte Login-Shell hängt vom Eintrag in */etc/passwd* ab. Für die CRAY im ZIB gilt, daß die *Bourne-Shell* voreingestellt ist. Darum wird hier die *Bourne-Shell* erläutert. Die Steuersprache der *C-Shell* ist ähnlich aufgebaut; sie hat einige zusätzliche Möglichkeiten aber auch Nachteile.

7.1.2 Das Shellskript

Ein Shellskript ist eine in einer Datei liegende zusammenhängende Kommandofolge, die eine Prozedur bildet. Zum Beispiel stehe folgendes in der Datei *laenge1*:

```
#Prozedur laenge1: berechnet Laenge der Datei myfile
echo 'Die Anzahl Woerter in myfile ist'
cat myfile | wc -w
```

Von # bis *Zeilenende* ist Kommentar. In einem Shellskript kann sich ein Kommando über mehrere Zeilen erstrecken; das Zeichen \ verhindert dabei eine Interpretation des jeweiligen Folgezeichens durch die Shell. Steht also ein \ unmittelbar vor einem *linefeed*, wird dieser unterdrückt und beide Zeilen als eine Zeile interpretiert. Will man hingegen im o.g. Beispiel Text und Längenangabe in einer Zeile erhalten, kann man mit

```
echo 'Die Anzahl Woerter in myfile ist \c'
```

das *linefeed* nach der Ausgabe unterdrücken.

Ausführung:

Eine Prozedur in *datei* kann auf zwei Weisen ausgeführt werden:

Methode 1:	sh *datei argumente*
Methode 2:	chmod u+x *datei*
	datei argumente

Wenn *datei* nicht im aktuellen Verzeichnis steht, muß man den Pfadnamen angeben (siehe Kapitel 7.7.2). Nach der 2. Methode wird *datei* ausführbar, sie bleibt es auch nach Veränderungen.In beiden Methoden können Prozeduren durch Anhängen von "&" auch im Hintergrund ausgeführt werden.

Beispiel: mit *laenge1* wie oben bringt

```
sh laenge1
```

folgende Ausgabe, wenn *myfile* 32 Wörter enthält:

```
Die Anzahl Woerter in myfile ist
      32
```

7.2 Parameter

7.2.1 Positionale Parameter

Eventuell nach dem Dateinamen angegebene Argumente werden positionalen Parametern zugewiesen, deren Werte mit *$1, $2, ...* anzusprechen sind. Nicht definierte Parameter werden mit dem *Nullstring* belegt.

Sämtliche positionale Parameter können auch innerhalb einer Prozedur mit

set *string1 string2 ...*

gesetzt bzw. umdefiniert werden.

Beispiel:

Die Datei *laenge2* enthalte folgendes:

```
#Prozedur laenge2: berechnet Laenge einer Datei
echo "Die Anzahl Woerter in $1 ist"
cat $1 | wc -w
```

Dem Kommando

```
sh laenge2 myfile
```

folgt wie vorher die Ausgabe

```
Die Anzahl Woerter in myfile ist
       32
```

Quoting:

Beachten Sie den Unterschied zwischen "..." und '...':

* innerhalb doppelter Anführungszeichen werden die Parameter durch ihre Werte ersetzt;
* innerhalb einfacher Anführungszeichen findet keine Ersetzung statt.

Das Kommando

```
echo 'Die Anzahl Woerter in $1 ist'
```

gäbe in *laenge2* nicht die gewünschte Ausgabe!

shift-Kommando:

Nur die positionalen Parameter 1 bis 9 können in der Prozedur direkt angesprochen werden. Weitere Parameter sind aber mit *shift* zu erreichen:

shift *n*

(Voreinstellung: n = 1) löscht die ersten *n* Argumente und weist den übrigen erneut die positionalen Parameter 1 bis 9 zu; z.B. würden mit

```
shift 8
echo $1 $3
```

die Werte der (ursprünglich) 9. und 11. Argumente ausgegeben.

7.2.2 Sonderparameter

Folgende Werte der Sonderparameter können auch in Shellskripten verwendet werden:

$0	der Name der Prozedur
$#	die Anzahl angegebener Argumente
$*	der Satz angegebener Argumente (als ein String)
$@	der Satz angegebener Argumente: falls innerhalb doppelter Anführungszeichen, als $# Strings; sonst als ein String
$$	die PID-(Prozeß-Identifikations-)Nummer des aktuellen Prozesses - z.B. nützlich, um einen eindeutigen Dateinamen zu erzeugen
$?	der "Exit-Status" der zuletzt ausgeführten Shell (siehe Kapitel 7.6.2)
$-	die Optionen, welche der Shell beim Aufruf oder durch *set*- Kommandos zugewiesen werden.

Beispiel: die Datei *druckel* enthalte folgendes:

```
#Prozedur druckel: gibt mehrere Dateien auf stdout aus
cat $*
echo "$# Datei(en) auf stdout ausgegeben"
```

Nach

```
sh druckel datei1 datei2 datei3
```

werden *datei1, datei2* und *datei3* konkateniert auf *stdout* ausgegeben, dazu die Meldung

```
3 Datei(en) auf stdout ausgegeben
```

7.2.3 Bedingte Ersetzung von Parametern

Es ist hilfreich, *$* als einen Ersetzungsoperator zu betrachten:

Ausdruck	**Ersetzung**
$*parameter*	Falls *parameter* definiert (d.h. angegeben) wurde, dessen Wert; sonst der Nullstring

Andere nützliche Ersetzungen sind folgende:

${*parameter*}*wort*	Inhalt von *parameter* wird mit *wort* verknüpft.
${*parameter*:-*wort*}	Falls *parameter* nicht definiert wurde oder

	$parameter der Nullstring ist, *wort*; sonst *$parameter*
${*parameter:+wort*}	Falls *parameter* definiert wurde, aber *$parameter* nicht der Nullstring ist, *wort*; sonst der Nullstring
${*parameter:?*}	Falls *parameter* nicht definiert wurde oder *$parameter* der Nullstring ist, Abbruch
${*parameter:?*meldung}	Falls *parameter* nicht definiert wurde oder *$parameter* der Nullstring ist, Abbruch mit Ausgabe von *meldung*

Beachten Sie, daß dabei *$parameter* nicht geändert wird!

Wird in einem dieser Ausdrücke der Doppelpunkt weggelassen, wird nur kontrolliert, ob der Parameter definiert bzw. nicht definiert wurde.

Die letzten zwei Ausdrücke erlauben es, eine Prozedur kontrolliert abzubrechen, falls ein erforderlicher Parameter nicht angegeben bzw. der Nullstring ist. Dazu verwendet man das Kommando *:* , das nichts anderes tut, als Variablen und Parameter zu ersetzen. Zum Beispiel kann man mit

: ${1?fehlt} ${2?fehlt}

am Anfang einer Prozedur kontrollieren, ob wenigstens zwei Parameter angegeben sind.

Beispiel:

```
#Prozedur Anzahl1:  gibt Anzahl Dateien in einem Ver-
                        zeichnis (Voreinstellung: $HOME) an
echo "Die Anzahl Dateien in ${1:-$HOME} ist"
ls ${1:-$HOME} | wc -l
```

Es sei angenommen, daß die Variable HOME exportiert wurde (siehe Kapitel 7.3.4)

7.3 Variablen

7.3.1 Variablen in Prozeduren

Variablen können in einem Shellskript ebenso verwendet werden wie in der Login-Shell. Die Wertzuweisung folgt mittels =, und der Wert ist mit *$* anzusprechen.

Für den Fall, daß eine erforderliche Variable nicht definiert bzw. der *Nullstring* ist, können, um sinnvoll zu verfahren, die in Kapitel 7.2.3 beschriebenen Ausdrücke auch mit Variablen verwendet werden. Hinzu kommt die Form

${*variable:=wort*}

die wie *${variable:-wort}* wirkt, aber zusätzlich *variable* den Wert *wort* zuweist. Auf diese Weise kann eine Voreinstellung der Variablen getroffen werden.

Jede Prozedur bzw. jedes Kommando (außer eingebauten Systemkommandos) startet normalerweise eine Subshell als Kindprozeß. Wenn nichts anderes getan wird, werden keine Variablenwerte von der aufrufenden Shell in die Subshell kopiert. Nach Ausführung des Kommandos werden ebenfalls keine Variablenwerte der aufrufenden Shell zurückgegeben (es wird lediglich ein Exit-Status zurückgegeben). Also sind per Voreinstellung die Werte gleichnamiger Variablen in der Login-Shell und in jeder Prozedur völlig voneinander unabhängig.

7.3.2 Bezug auf Kommandoergebnisse

Einer Variablen wird das Ergebnis eines Kommandos zugewiesen, wenn dieses innerhalb umgekehrter Hochkommas `` `...` `` (auch als *accents graves* bezeichnet) steht; z.B. weist

```
katalog=`pwd`
```

der Variablen *katalog* den Namen des aktuellen Verzeichnisses zu.

Innerhalb `` `...` `` werden Parameter und Variablen durch ihre Werte ersetzt, wenn sie mit vorangestelltem *$* angegeben sind; Muster für Dateinamen werden expandiert.

Beispiel: Die Datei *laenge3* enthalte folgendes:

```
#Prozedur laenge3: Verbesserung von laenge2
Anzahl=`cat $1 | wc -w`
echo "Die Anzahl Woerter in $1 ist $Anzahl."
```

Dann erzeugt das Kommando

```
sh laenge3 myfile
```

die Ausgabe

```
Die Anzahl Woerter in myfile ist         32.
```

7.3.3 Interpretation einer Kommandozeile

Eine Kommandozeile kann geschachtelte Metazeichen enthalten. Dann ist es wichtig zu verstehen, in welcher Reihenfolge die Interpretation erfolgt.

In einer Kommandozeile sollten Anführungszeichen jeden Typs *('...'; "..."; `...`)* in gerader Zahl vorkommen. Dann ist der String zwischen den ersten zwei Anführungszeichen derselben Art gequotet, der String bis zum nächsten Anführungszeichen ist nicht gequotet, der String zwischen diesem Anführungszeichen und den nächstfolgenden Anführungszeichen derselben Art ist gequotet, u.s.w. Um eine eindeutige Quotingsschachtelung zu erreichen, sollte auch zwischen jedem Paar von Anführungszeichen eines Typs eine gerade Zahl Anführungszeichen jedes anderen Typs vorkommen.

Als erster Interpretationsschritt werden Syntaxzeichen interpretiert, die nicht gequotet bzw. durch \ unterdrückt werden. Die Syntaxzeichen sind

```
>  >>  <  <<  ()  {}  |  &  ;
```

Daraus ergibt sich eine Folge zu interpretierender Kommandos sowie Regeln, wie deren Standardeingaben und Standardausgaben miteinander verknüpft werden sollen.

Die Kommandointerpretation folgt von außen nach innen: Zuerst wird jeder String, der auf die oben geschriebene Weise gebildet wurde, nach den entsprechenden Quotingregeln interpretiert. Enthält ein String weitere Quotes, wird jeder so gebildete Substring interpretiert und dieses Verfahren wiederholt, bis alles interpretiert ist.

Wir fassen die Quotingsregeln für Standard-UNIX zusammen. Beachten Sie, daß (mit einer Ausnahme) die Bedeutung jedes Sonderzeichens mit einem vorangestellten \ unterdrückt werden kann.

Einfache Anführungszeichen ('...')

* keine Ersetzung

Die Ausnahme: '\' innerhalb '...' bewirkt nicht, daß die Sonderbedeutung eines Hochkommas unterdrückt wird. Ein String, der Hochkommas enthält, muß mit doppelten Hochkommas spezifiziert werden.

Doppelte Anführungszeichen ("...")

* eingeschlossene gequotete Strings werden zuerst nach ihren Regeln interpretiert
* sonst werden Variablen und Parameter mit vorangestelltem *$* durch ihre Werte ersetzt.

Umgekehrte Anführungszeichen (` ... `)

* eingeschlossene gequotete Strings werden zuerst nach ihren Regeln interpretiert
* sonst werden Variablen und Parameter mit vorangestelltem *$* durch ihre Werte ersetzt, und
* Dateinamen werden nach den mit *, ? und [...] angegebenen Mustern generiert; falls ein Muster keiner Datei entspricht, wird das Muster nicht expandiert, sondern buchstäblich weitergegeben.

Anmerkung: Syntaxzeichen werden innerhalb von Quotes nie interpretiert!

Beispiele: Es sei angenommen, das aktuelle Verzeichnis enthalte unter anderem die Dateien *a3, p1.a* und *p2.a*, die aktuelle Umgebung enthalte eine Variable *x* mit Wert *a3*.

Eingabe	Interpretation	Ergebnis
`ls *.a $x >d`	`ls [p1.a p2.a a3] >d`	Die Namen *p1.a, p2.a, a3* werden in die Datei *d* geschrieben
`'ls *.a $x >d'`	`ls [*.a $x >d]`	Die Namen **.a, $x, >d* werden am Bildschirm ausgegeben
`'ls *.a $x' >d`	`ls [*.a $x] >d`	Die Namen **.a, $x* werden in die Datei *d* geschrieben
`"ls *.a $x"`	`ls *.a a3`	Die Namen **.a* und *a3* werden am Bildschirm ausgegeben
`` `ls *.a $x` ``	`p1.a p2.a a3`	Fehlermeldung, wenn *p1.a* nicht ausführbar ist; sonst das Ergebnis dieses Kommandos mit den Argumenten *p2.a* und *a3*
`` `ls "*.a $x"` ``	`*.a a3`	Fehlermeldung, wenn **.a* nicht ausführbar ist; sonst das Ergebnis dieses Kommandos mit dem Argument *a3*

(In obigen Beispielen wird eine Fehlermeldung ausgegeben, falls **.a*, *$x* oder *>d* nicht existiert. Außerdem wird, falls eine dieser Dateien ein Verzeichnis ist, anstatt des Namens eine Liste der enthaltenen Dateien ausgegeben.)

7.3.4 Wertübergabe an Prozeduren

Gemeinsame Variablen:

Um Variablen in einer Prozedur in *datei* gleichnamigen Variablen in einer aufrufenden Prozedur gleichzustellen, kann die Prozedur mit

```
. datei
```

aufgerufen werden. Damit läuft die aufgerufene Prozedur in derselben Shell wie die aufrufende Prozedur, so daß mit denselben Namen dieselben Variablen angesprochen werden. Es können daher keine Argumente angegeben werden.

Zum Beispiel enthalte *setup* Kommandos zur Erzeugung einer besonderen Umgebung mit gewissen Variablen. Mit . *setup* wird diese Umgebung in der Login-Shell kreiert und die Variablen werden zur Verfügung gestellt.

Übergeben von Variablenwerten:

Es gibt zwei Methoden, einen Variablenwert an eine Subshell zu übergeben, ohne daß Veränderungen in die aufgerufene Prozedur zurückübertragen werden:

1) Mit

export *variablenliste*

wird für jede Variable in der Liste ein Attribut gesetzt, das bestimmt, daß der Wert dieser Variablen allen aufgerufenen Prozeduren (und allen deren Kind- bzw. Nachfolgerprozessen) übergeben werden soll. Dieses Attribut bleibt erhalten, bis die aufrufende Shell endet - unabhängig davon, wie häufig die Variable umdefiniert wird.

2) Mit

name1 = wert1 name2 = wert2 ... datei

werden in der aufrufenden Shell den Variablen *name1, name2, ...* die Werte *wert1, wert2, ...* zugewiesen und die Prozedur in *datei* mit diesen Werten als Voreinstellungen ausgeführt. In diesem Fall heißen *name1, name2, ...* Schlüsselwortparameter.

Rückgabe von Variablenwerten

Ein in der aufgerufenen Prozedur eingestellter oder berechneter Variablenwert kann der aufrufenden Shell mit *echo* und einer Kommandoersetzung zurückgegeben werden.

Beispiel: die Benutzerprozedur *nummer* berechne die Variable *zahl1* und gebe auf die Standardausgabe nur den Wert von *zahl1* vermittels

```
echo $zahl1
```

aus. Wenn nun die aufrufende Shell die Anweisung

```
zahl2=`nummer`
```

enthält, wird der Wert von *zahl1* in *nummer* der Variablen *zahl2* in der aufrufenden Shell zugewiesen.

7.4 Bereitstellen von Daten für eine Prozedur

Einlesen von einer Datei unter *stdin*

Für eine Prozedur benötigte Daten können von einer Datei unter *stdin* mit *read* gelesen werden.

Beispiele: nach

```
read a b c  < datei
```

wird die erste Zeile von *datei* gelesen. Das erste Wort wird der Variablen *a*, das zweite der Variablen *b* und der Rest der Variablen *c* zugewiesen.

Mit

```
{read a; read b;}  < datei
```

werden die ersten zwei Zeilen gelesen.

here documents

Daten für ein Kommando können in einer Prozedur mit einem sogenannten *here document* bereitgestellt werden (siehe Kap. 2.4).

Ersetzungen finden statt innerhalb eines *here document* wie in Kapitel 7.3.3 beschrieben. Die Wirkung von einzelnen *$* und * kann durch Voranstellung von \ unterdrückt werden. Ersetzungen im ganzen *here document* können verhindert werden, wenn der erste Trennstring in ' (single quotes) gesetzt wird:

```
cat > datei << '%'
.
... der $- Preis ist ...
.
%
```

7.5 Ablaufsteuerung

7.5.1 Die for-Anweisung

Um eine bestimmte Anzahl von Wiederholungen einer Kommandofolge zu erreichen, gibt es die *for*-Anweisung, und zwar in zwei Formen:

1. for *variable*
   do
   kommandoliste
   done

   kommandoliste wird *$#* mal ausgeführt, in jeder Schleife wird *variable* durch eines der angegebenen Argumente *$** (in aufsteigender Reihenfolge) ersetzt.

2. for *variable* in *wörterliste*
   do
   kommandoliste
   done

kommandoliste wird für jedes Wort in *wörterliste* einmal ausgeführt, in jeder Schleife wird *variable* durch ein Wort in *wörterliste* (in derselben Reihenfolge) ersetzt. Die 1. Form ist eigentlich eine Abkürzung für die 2. Form, wenn *wörterliste* gleich *$** ist.

Die reservierten Wörter *for, do* und *done* müssen jeweils entweder auf einer neuen Zeile stehen oder einem ; folgen; z.B. kann die erste Form oben auch geschrieben werden als

for *variable*; do *kommandoliste*; done

Metazeichen werden interpretiert wie in Kapitel 7.3.3 beschrieben.

Beispiel: Die Datei *laenge4* enthalte folgende Kommandos:

```
#Prozedur laenge4: berechnet die Laengen von mehreren Dateien
for file
do
      Anzahl=`cat $file | wc -w`
      echo "Die Anzahl Woerter in $file ist $Anzahl."
done
```

Dann gibt das Kommando

```
sh laenge4 datei1 datei2 datei3
```

die Längen der drei Dateien *datei1, datei2* und *datei3* nacheinander aus.

7.5.2 Die case-Anweisung

Zur Verzweigung je nach Wert einer Variablen gibt es die *case-* Anweisung. Syntax:

case *$variable* in
   muster1) kommandoliste1 ;;
   muster2) kommandoliste2 ;;
   ...
   musterN) kommandolisteN ;;
esac

Jedes Muster kann ein einzelnes Wort sein oder mit den bekannten Metazeichen gebildet werden.

Beispiele:

`1)`	der Wert 1
`[abc])`	die Werte a, b oder c
`*.doc)`	Dateien mit Erweiterung .doc
`*)`	alle noch nicht erwähnten Werte
`\*)`	der Wert *

sowie auch:

`ab|cde)`	entweder ab oder cde

Beispiel:

```
#Prozedur laenge5:      berechnet die Laengen von mehreren
#                       Dateien, entweder einzeln (1. Parameter
#                       = -e) oder zusammen (1. Parameter = -z)
case $1 in
       -e)  shift       # ersten Parameter ueberspringen
            for file
            do
                  Anzahl=`cat $file | wc -w`
                  echo "Die Anzahl Woerter in $file ist $Anzahl."
            done;;
       -z)  shift       # ersten Parameter ueberspringen
            Anzahl= `cat $* | wc -w`
            echo "Die Gesamtanzahl Woerter in den Dateien"
            echo "$*"
            echo "ist $Anzahl.";;
        *)  echo "usage: laenge5 -[ez] Dateiliste";;
esac
```

7.5.3 Bedingungen

Zum Testen, ob eine Bedingung stimmt, dient das Kommando *test*, abgekürzt *[...]* (Leerstelle nach der ersten und vor der zweiten eckigen Klammer erforderlich). Das Kommando hat zwar keine Ausgabe, jedoch einen Exit-Status (0 = wahr, sonst = falsch).

Beispiele:

Kommando	**ist wahr, wenn:**
[-d *name*]	Verzeichnis *name* existiert
[-f *name*]	Datei *name* existiert
[-s *name*]	Datei *name* existiert und nicht leer ist
[-r *name*]	Datei *name* existiert und lesbar ist
[-w *name*]	Datei *name* existiert und schreibbar ist
[-x *name*]	Datei *name* existiert und ausführbar ist
[*string*]	*string* nicht der Nullstring ist
[*name1* = *name2*]	Strings *name1* und *name2* identisch sind
[*zahl1* -eq *zahl2*]	Zahlen *zahl1* und *zahl2* gleich sind
	(analog mit *-ne*, *-gt*, *-ge*, *-lt*, *-le*)

Verknüpfungen Verknüpfungen können mit *-a* (AND), *-o* (OR) und *!* (NOT) gemacht werden.

Beispiele:

[-d *name1* -a -f *name2*]	ist wahr, wenn Verzeichnis *name1* und Datei *name2* existieren
[! -r *name*]	ist wahr, wenn Datei *name* entweder nicht existiert oder nicht lesbar ist

Besonderheit:
[*string1* != *string2*] hat dieselbe Bedeutung wie [! *string1* = *string2*].

7.5.4 Die if-Anweisung

Verzweigung mit *if* gibt es in zwei Formen:

1. if *bedingung*
   then
   kommandoliste
   fi

2. if *bedingung*
   then
   kommandoliste1
   else
   kommandoliste2
   fi

Beide Formen können geschachtelt werden. Auch kann *else if* mit *elif* abgekürzt werden, wobei dann nur ein *fi* am Ende benötigt wird (s. Beispiel unten).

Die reservierten Wörter *if, then, else, fi* und *elif* müssen jeweils entweder auf einer neuen Zeile stehen oder einem ";" folgen.

Beispiel:

```
#Prozedur drucke3: Verbesserung von drucke1
for file
do
      if [ -d $file ]
      then
            echo "$file ist ein Verzeichnis"
      elif [ ! -s $file ]
      then
            echo "$file ist leer"
      else
            echo "$file wird auf stdout ausgegeben"
            cat $file
      fi
done
```

7.5.5 Die while- und until-Anweisungen

Durch eine Kommandofolge mit Syntax

while *bedingung*
do
kommandoliste
done

wird *kommandoliste* wiederholt ausgeführt, solange *bedingung* wahr ist. Mit

until *bedingung*
do
kommandoliste
done

wird wiederholt *kommandoliste* ausgeführt, solange *bedingung* falsch ist.

Die reservierten Wörter *while, until, do* und *done* müssen jeweils entweder auf einer neuen Zeile stehen oder einem ";" folgen.

Beispiel:

```
#Prozedur laenge6: Alternativ zu laenge4
while [ $1 ]
do
      Anzahl=`cat $1 | wc -w`
      echo "Die Anzahl Woerter in $1 ist $Anzahl."
      shift
done
```

7.6 Fehlerbehandlung

7.6.1 Ablaufprotokollierung

Es gibt zwei Möglichkeiten, den Ablauf der Prozeduren in *stderr* zu verfolgen:

1. Mit

   sh -v datei argumente

wird jedes ausgeführte Kommando buchstäblich (d.h. ohne Ersetzung von Parametern u.s.w.) im Ablaufprotokoll angezeigt, gefolgt von einer eventuellen Ausgabe.

2. Mit

   sh -x datei argumente

wird jedes Kommando interpretiert (d.h. nach Ersetzungen) ausgegeben, gefolgt von einer eventuellen Ausgabe. Die interpretierten Kommandos werden mit + markiert.

Die zwei Möglichkeiten können mit

   sh -vx datei argumente

kombiniert werden.

Die Flags *-v* und *-x* können auch in der Prozedur selber mit *set -v, set -x* bzw. *set -vx* eingestellt werden. Jeder Aufruf dieser Prozedur würde aber einzeln protokolliert ablaufen.

Beispiel:

Mit der Prozedur *druckel* im Kapitel 7.2.2 ergibt

```
sh -x druckel datei1 datei2 datei3
```

folgende Ausgabe:

```
+ cat Datei1 Datei2 Datei3
+ echo 3 Datei(en) auf stdout ausgegeben
3 Datei(en) auf stdout ausgegeben
```

7.6.2 Der Exit-Status

Jedes Kommando gibt der aufrufenden Shell einen Exit-Status zurück. Er ist 0, falls das Kommando normal endet. Für die meisten Kommandos kann er in zwei Fällen anders als 0 sein:

* Das Kommando konnte ausgeführt werden, gab aber ein Nullergebnis; entweder beim Suchen oder beim Testen, ob eine Bedingung wahr ist.

* Das Kommando konnte nicht ausgeführt werden.

In beiden Fällen kann der Exit-Status in *if-* bzw. *&&-* oder ||-Anweisungen verwendet werden, um nachfolgende Schritte zu bestimmen.

Der Exit-Status des zuletzt ausgeführten Kommandos ist im Sonderparameter *$?* gespeichert.

Beispiele:

`[ -f datei ] && cat datei`	schreibt *datei* nach *stdout*, falls sie existiert
`cat datei1 || cat datei2`	schreibt *datei1* nach *stdout*, falls sie existiert; sonst *datei2*

Bemerkung:

Wenn ein Kommando mit einem Exit-Status anders als *0* endet, wird bei einem normalen Aufruf in folgenden Fällen einfach fortgesetzt:

* alle Fälle vom ersten Typ
* Fehler in der Umleitung der Ausgabe bzw. Eingabe
* "abnormal termination" des Kommandos

Bei allen anderen Fällen vom zweiten Typ wird die Prozedur abgebrochen.

Wenn die Prozedur aber mit

sh -e name

aufgerufen wird, wird nach dem ersten Kommando, das mit einem Exit-Status anders als 0 endet, immer abgebrochen. Innerhalb einer Prozedur kann man auch mit *set -e* und später *set +e* erreichen, daß innerhalb eines bestimmten Bereichs ein Abbruch folgt.

Bemerkung: Im ZIB wird der eigentliche CRAY-Batchjob mit *sh -ex* aufgerufen (vergleiche ZIB-Standardprolog, Kapitel 3.5), also mit Einzelprotokollierung und einem Abbrechen des Jobs beim ersten fehlerhaften Kommando.

7.7 Zugriff auf Prozeduren

7.7.1 Shellfunktionen

Eine durch ein Shellskript definierte Prozedur kann von der Login-Shell und von allen Subshells aufgerufen werden, sie muß jedesmal von der Platte geholt werden und läuft in ihrer eigenen Subshell ab. Man kann eine Prozedur aber auch so definieren, daß sie analog zu eingebauten Systemkommandos im Hauptspeicher liegt und in der aktuellen Shell oder Subshell läuft. (Eine solche Prozedur kann aber nur in der Shell aufgerufen werden, in der sie definiert wird). Das bewirkt eine Kommandofolge nach folgendem Muster:

```
name(){
        kommandoliste
}
```

Eine solche Prozedur heißt eine Shellfunktion. Der Aufruf erfolgt mit *name argumente*. Wenn man eine Shellfunktion in *.profile* definiert, wird die Prozedur in der Login-Shell verfügbar. Das *set*–Kommando gibt an, welche Shellfunktionen aktuell definiert sind.

Eine Shellfunktion kann mit

unset *name*

gelöscht werden.

Eine Shellfunktion kann man auch in einem Shellskript definieren, wenn zum Beispiel dieselbe Funktion mehrmals verwendet wird.

Beachten Sie, daß eine neu definierte Shellfunktion oder eine neu definierte Variable eine schon definierte, gleichnamige Shellfunktion oder Variable überschreibt.

7.7.2 Suchfolge für Prozeduren

Es empfiehlt sich, alle getesteten Benutzerprozeduren in ein Verzeichnis, etwa *$HOME/bin*, zu verlagern und die *PATH*-Variable etwa mit

PATH = /sbin:/bin:/usr/bin:/usr/ucb:/usr/lbin:.

in *.profile* umzudefinieren. Dabei verläuft die Suchfolge nach Kommandos folgendermaßen:

die eingebauten Systemkommandos
die vom Benutzer definierten Shellfunktionen
die vom Benutzer definierten Prozeduren im aktuellen Verzeichnis
die Systemkommandos in /sbin, */bin*, */usr/bin*, */usr/ucb*, */usr/lbin*
die Benutzerprozeduren in *$HOME/bin*.

Diese Suchfolge ist selbstverständlich nur dann zu beachten, wenn zwei Kommandos, Shellfunktionen bzw. Prozeduren denselben Namen haben. Wenn man trotzdem eine Prozedur aufrufen will, die sonst nicht erreicht würde, kann man das meistens tun:

* wenn die gewünschte Prozedur *name* auch ein eingebautes Systemkommando bzw. eine Shellfunktion ist, mit *sh name* oder durch Voranstellung des Verzeichnisnamens (etwa */usr/ucb*, */bin*, */usr/bin* oder *$HOME/bin*)

* wenn die gewünschte Prozedur auch ein nicht eingebautes Systemkommando bzw. eine Benutzerprozedur ist, durch Voranstellung des Verzeichnisnamens.

Eine Shellfunktion, die denselben Namen hat wie ein eingebautes Systemkommando, kann nie aufgerufen werden.

8. FORTRAN Übersetzer-Optionen und -Direktiven

Die Fa. CRAY bietet mit CFT77 und CFT zwei FORTRAN-Übersetzer an, die dem Standard ANSI X 3.9-1978 (FORTRAN77) genügen. Beide werden im folgenden detailliert beschrieben, und zwar die Versionen CFT77 3.1 und CFT 1.16. CFT hat mit dieser Version das Ende seiner Entwicklung erreicht; er wird ab 1.1.1991 nur noch im "maintenance mode" angeboten, d.h. er steht dann noch solange zur Verfügung, wie die zu diesem Zeitpunkt freigegebene Version des Betriebssystems UNICOS (voraussichtlich 6.0) am ZIB im Einsatz ist. CFT77 hat inzwischen einen mit CFT vergleichbaren Leistungsumfang und generiert zum Teil deutlich schnelleren Code. CFT77 sollte also in Zukunft vorzugsweise eingesetzt werden.

Zusätzlich zu den FORTRAN77-Übersetzern CFT77 und CFT gibt es zwei weitere Übersetzer, den FORTRAN-Präprozessor *fpp* und den FORTRAN-Multitasking-Übersetzer *fmp*. Eine Einführung in deren Anwendung gibt Kapitel 8.5.

Der einfachste Aufruf der FORTRAN77-Übersetzer CFT77 bzw. CFT unter Benutzung der Standardwerte für die beteiligten Dateien lautet (siehe 4.2.1 FORTRAN Übersetzer-Aufrufe (CFT77 und CFT)) z.B.:

```
cft77 -es quelle.f    bzw.    cft quelle.f
```

Arbeitsweise und Leistungen der Übersetzer können einerseits durch Optionen beim Aufruf der Übersetzer und andererseits durch Direktiven in der Programmquelle gesteuert werden.

Übersetzer-Optionen steuern das globale Verhalten für einen Übersetzerlauf wie Ein-/Ausgabe der Übersetzer, Optimierung, insbesondere Vektorisierung der Programme, Hilfen zur Überwachung und Fehlersuche sowie Varianten zur Kompatibilität von Programmen. Bei den Parameterwerten ist jeweils die Voreinstellung (default) unterstrichen!

Übersetzer-Direktiven passen das Verhalten der Übersetzer lokalen Besonderheiten im Programm an.

8.1 Übersetzer-Optionen für CFT77

Es gibt im wesentlichen zwei verschiedene Arten, um Übersetzer-Optionen zu setzen:

Explizite Angabe in der Form

-option wert

Beispiel:

```
-i 64              bei einem Wert oder
-o full,zeroinc    bei mehreren Werten.
```

oder Ein- bzw. Ausschalten einer Reihe von Optionen in der Form

-e onstring bzw. -d offstring

Beispiel:

```
-e Afx    oder    -d p
```

Dabei bezeichnet jeder nach *-e* bzw. *-d* spezifizierte Buchstabe eine Option.

Ein Übersetzeraufruf mit dem FORTRAN-Programm in der Datei *quelle.f* kann dann bei Benutzung von Optionen so aussehen:

Beispiel:

```
cft77 -d p -e Afx -i 64 -o full,zeroinc quelle.f
```

8.1.1 Optionen zur Steuerung der Ein-/Ausgabe

Allgemeine Annahme: Das FORTRAN-Programm liege in der Datei *quelle.f*.

quelle.f	Name der Datei, in der die Quelle steht; sollte mit *.f* enden.
-l listfile quelle.l	*listfile* bezeichnet den Namen der Datei, in die die Übersetzerliste geschrieben wird. Die Liste wird nur erzeugt, wenn *-e* mit *L, S, g, h, m, s* oder *x* gesetzt ist (s.u.).
-b binfile quelle.o	*binfile* bezeichnet den Namen der Datei, in die der Übersetzer die binären Lademodule schreibt.

-e onstring bzw. -d offstring:

-e/-d g	Gibt den erzeugten Code für jede Programmeinheit in die Datei *quelle.l* bzw. in die durch *-l* festgelegte Datei aus (siehe 8.2.1 Direktiven CODE und NOCODE).
-e/-dh	*Head line*: Bewirkt, daß nur die erste Zeile jeder Programmeinheit sowie Fehlermeldungen auf *stderr* ausgegeben werden; soll stattdessen in eine Datei ausgegeben werden, so muß dies explizit mit der Option *-l* festgelegt werden (kein Default!).
-e/-d m	Wirkt wie *-es*, nur werden zusätzlich alle DO-Schleifen in der Ausgabeliste durch Buchstaben geklammert, die den Grad der Vektorisierung kennzeichnen. Der Parameter *s* darf nicht gleichzeitig verwendet werden.
-e/-d s	*Source*: Gibt die FORTRAN-Quelle in die Datei *quelle.l* bzw. in die durch *-l* festgelegte Datei aus.
-e/-d x	*Cross reference*: erzeugt eine Querverweis-Tabelle aller Symbole einer Programmeinheit.
-e/-d B	Bewirkt, daß ein binärer Lademodul erzeugt wird. *-dB* unterdrückt also die Generierung.
-e/-d L	*Listings*: *-eL* wirkt wie *-e gsx* (s.o.), d.h. es werden alle entsprechenden Listen in die Datei *quelle.l* oder die durch *-l* festgelegte Datei ausgegeben.
-e/-d S quelle.s	Es wird eine Datei erzeugt, in der Pseudo-Code für den CRAY-Assembler (CAL) abgelegt wird. Der erzeugte Text kann nach wenigen Handkorrekturen als Eingabe für CAL dienen. DATA-Anweisungen werden nicht unterstützt. Durch *-s file* kann diese Ausgabe in eine Datei mit Namen *file* gelenkt werden.

8.1.2 Optionen zur Vektorisierung und Optimierung

-o optim	Folgende Werte für *optim* sind erlaubt (werden mehrere Werte spezifiziert, so sind sie durch Kommata zu trennen):
-o nozeroinc zeroinc	Mit diesen beiden Optionen wird CFT77 mitgeteilt, ob *Increment Variables* (CIV siehe 12.4.2) mit einer Variablen vom Wert Null inkrementiert werden können oder nicht. Bei *zeroinc* geht der Übersetzer davon aus, daß in der Form CIV = CIV + variable die Variable auch den Wert Null haben kann und daher nicht vektorisiert werden darf.
-o full novector off	Mit diesen drei Optionen werden Optimierung und Vektorisierung des CFT77 gesteuert. Es werden bei der Codegenerierung alle nur möglichen Optimierungen und Vektorisierungen (*full*), nur skalare Optimierung (*novector*) oder gar keine Optimierung/Vektorisierung (*off*) durchgeführt.

-I inlinefile

Aktiviert die Erzeugung von *inline*-Code: Aufrufe der in *inlinefile* enthaltenen Unterprogramme werden unter gewissen Voraussetzungen durch die Unterprogramme selbst ersetzt; dadurch entfallen Unterprogrammaufrufe, und DO-Schleifen mit Unterprogrammaufrufen werden dadurch u.U. vektorisierbar. *inlinefile* kann die gleiche Datei wie *quelle.f* sein; in diesem Fall muß die Datei ein mit der PROGRAM-Anweisung beginnendes Hauptprogramm enthalten.

Anmerkungen: Bei der Auswahl der zu expandierenden Programme ist zu bedenken, daß je nach Anzahl der Aufrufe und Größe der Unterprogramme der Umfang des erzeugten Codes stark zunehmen kann.
Das Einfügen der Unterprogramme in aufrufende Programmeinheiten geschieht im Gegensatz zu *fpp* nicht in FORTRAN (und nicht sichtbar für den Benutzer), sondern in einem vom Übersetzer erzeugten Zwischencode.

8.1.3 Optionen zur Überwachung und Fehlersuche

-m n — *Message level*: Gibt das niedrigste Niveau der Meldungen an, die CFT77 ausgeben soll.
-m 0 — *Comment*: Meldungen aller Stufen werden ausgegeben.
-m 1 — *Note*: Kommentare zur Effizienz des Programms werden unterdrückt.
-m 2 — *Caution*: Zusätzlich werden Anmerkungen über mögliche Schwierigkeiten mit anderen Übersetzern (z.B. nicht ANSI) unterdrückt.
-m 3 — *Warning*: Zusätzlich werden Hinweise auf mögliche Fehler unterdrückt (z.B. eine Anweisung, die nicht durchlaufen wird).
-m 4 — *Error*: Zusätzlich werden Warnungen vor wahrscheinlichen Fehlern (z.B. Benutzung eines Feldes mit zu wenig Indizes) unterdrückt. Fehler (Niveau 4) können nicht unterdrückt werden.

-e onstring bzw. -d offstring:

-e/-d a — *Abort*: Der Job wird nach der Übersetzung abgebrochen, falls ein fataler Fehler bei der Übersetzung aufgetreten ist. Ist *-d a* gesetzt (Standard), so werden nachfolgende Verarbeitungsschritte (z.B. die Programmausführung) auch dann ausgeführt, wenn schwerwiegende Fehler bei der Übersetzung aufgetreten sind.

-e/-d f — *Flowtrace*: Schaltet die Messung des Zeitverbrauchs, aufgeschlüsselt nach den einzelnen Programm-Modulen, ein bzw. aus. Dies ist eine für Optimierungszwecke sehr nützliche Option, um die rechenzeitintensiven Programmteile zu ermitteln. Sie sollte jedoch nur für einzelne Programmläufe eingesetzt werden, da sie im allgemeinen zu einem erheblichen zusätzlichen Zeitbedarf führt (in Abhängigkeit von der Anzahl der Unterprogrammaufrufe während der Laufzeit). Die Flowtrace-Option hat Vorrang vor den Übersetzer-Direktiven FLOW und NOFLOW (siehe Kapitel 8.2.3).
Anmerkung: Voraussetzung für die Verwendung der FLOWTRACE-Option ist das Ersetzen von CALL EXIT bzw. CALL ABORT durch STOP oder END. Die Ausgabe der erzeugten Information erfolgt nach *stdout*.

-e/-d o — *Array bounds checking*: Schaltet die Überprüfung von Feldindizes auf Bereichsüberschreitung ein bzw. aus. Diese Option verhindert nicht die Vektorisierung, verbraucht jedoch in der Regel sehr viel CPU-Zeit. Sie sollte nur in Einzelfällen benutzt werden und hat Vorrang vor den Übersetzer-Direktiven BOUNDS und NOBOUNDS.
Anmerkung: Felder innerhalb einer formatierten WRITE-Anweisung werden nicht überprüft; ferner entfällt die Prüfung der letzten Dimension eines Feldes, wenn diese mit * angegeben wird.

-e/-d q — Die Übersetzung bricht nach 100 Fehlern ab.

-e/-d z — *Debug symbol table*: Wirkt wie Parameterwert D, jedoch ist volle Optimierung/Vektorisierung möglich.
Anmerkung: Im Zusammenhang mit der *-o full*-Option kann es vorkommen, daß die zum Debuggen erforderliche Information für DO-Schleifen, die der Übersetzer nicht vektorisieren konnte, nicht erzeugt wird. Dies wird in der Übersetzer-Liste unter VECTORIZATION INFORMATION angezeigt. Abhilfe: die entsprechenden DO-Schleifen gezielt von der Vektorisierung ausschließen (siehe Kapitel 8.2.2).

-e/-d D — *Debug symbol table*: Erzeugt eine Debug-Symboltabelle zur Fehlersuche. Diese Option verbraucht während der Ausführung weder nennenswert CPU-Zeit noch Speicherplatz; allerdings sollte die Optimierung ausgeschaltet sein (- o off), da sonst der Zusammenhang zwischen FORTRAN-Anweisungen und generiertem Code u.U. schwer zu erkennen ist.

8.1.4 Optionen zur Programmportabilität

-C cputyp, cpuchar — Die Option kennzeichnet den Rechnertyp und mögliche Eigenschaften der Zielmaschine, auf der der erzeugte Code laufen soll.
Wenn die Option nicht gesetzt ist, wird Code für die Maschine erzeugt, auf der CFT77 läuft, beziehungsweise für die, die durch die UNICOS-Anweisung *target* festgelegt wurde. *cputyp* wird vom System gesetzt und sollte nur im Falle einer Crosscompilation angegeben werden. Syntax ohne Angaben von *cputyp*: *-C,cpuchar*. Mehrere Angaben zur Hardware werden durch Kommata getrennt.
Ohne Angabe von *cputyp* nimmt CFT77 die folgenden Eigenschaften der im ZIB installierten Anlage an:

cray-xmp — Maschinentyp.

noema — *Extended memory addressing*: Keine erweiterte Speicheradressierung.

nocigs — *Compressed index / gather scatter*: Keine Hardware für komprimierte Indizes und Gather-, Scatter-Operationen.

vpop — *Vector population*: Vektoreinheit zum Zählen gesetzter Bits vorhanden.

-e onstring bzw. -d offstring:

-e/-d j — Bewirkt, daß alle DO-Schleifen, abweichend vom Standard, mindestens einmal durchlaufen werden (sinnvoll bei der Umstellung von FORTRAN-IV auf FORTRAN 77).

-e/-d p — Ermöglicht oder verhindert DOUBLE PRECISION-Arithmetik.
-d p bewirkt folgendes:
- Alle DOUBLE PRECISION-Deklarationen werden wie REAL-Deklarationen behandelt.
- Doppelt genaue Funktionen werden in die entsprechenden einfach genauen Versionen umgewandelt.
- Doppelt genaue Konstanten werden in einfach genaue umgewandelt.
- D-Formate werden in E-Formate umgewandelt.
Diese Option ist für Programme sinnvoll, die an Anlagen entstanden sind, an denen wegen geringerer Maschinengenauigkeit mit DOUBLE PRECISION gerechnet wurde. An der CRAY-Anlage ist DOUBLE PRECISION wegen der hohen Genauigkeit (64 Bit Worte für einfache Genauigkeit) im allgemeinen nicht nötig. Wird es trotzdem eingesetzt, führt dies zu erheblicher Verlangsamung des Programmablaufs.

-e/-d A — Alle Anweisungen, die nicht dem ANSI-Standard entsprechen, werden gemeldet.

8.1.5 Sonstige Optionen

-t n	Zahl der Bits zwischen 0 und 47, die bei Gleitpunktresultaten abgeschnitten, d.h. auf 0
0	gesetzt werden sollen.
-i intlen	Länge von INTEGER-Zahlen in Bits.
46	Anmerkung: 46-Bit-Arithmetik ist schneller, 64-Bit-Arithmetik erlaubt größere Werte.
64	Eine Überschreitung des Zahlbereiches wird bei 46-Bit-Arithmetik nicht erkannt.
-a alloc	Bestimmt die Art, wie Variable im Speicher abzulegen sind.
static	Allen Variablen wird Speicher fest zugewiesen. Die Werte lokaler Variablen werden jedoch nicht über Aufrufe hinweg erhalten (es sei denn, die SAVE-Anweisung wird verwendet).
stack	Konstanten und solchen Variablen, die in DATA-, SAVE- oder COMMON-Anweisungen vorkommen, wird Speicher fest zugewiesen. Alle anderen Variablen werden dynamisch in einem Keller ("stack") verwaltet.
-V	Allgemeine statistische Angaben des CFT77 werden nach *stderr* und *quelle.l* bzw. in die durch *-l* spezifizierte Datei ausgegeben (default: kein Logfile).

-e onstring bzw. -d offstring:

-e/-d r	*Round*: Multiplikationsergebnisse werden gerundet.
-e/-d I	Besetzt den Kellerbereich so vor, daß Gleitpunktoperationen mit undefinierten REAL-Variablen und Feldzugriffe mit undefinierten INTEGER-Variablen zu Fehlermeldungen führen.

8.2 Übersetzer-Direktiven für CFT77

Übersetzer-Direktiven erlauben es einerseits, einige Übersetzer-Optionen für bestimmte Programmabschnitte ein- bzw. auszuschalten und eröffnen andererseits zusätzliche Möglichkeiten. Sie werden in der Form

CDIR$ value

ins FORTRAN Quellprogramm eingestreut. Sie beginnen stets ab Spalte 1 und werden daher von anderen Übersetzern wie Kommentare behandelt; sie stören die Portabilität der Programme nicht.

Grundsätzlich haben Angaben im CFT77-Kommando gegenüber Übersetzer-Direktiven höhere Priorität. Alle Direktiven außer EJECT, LIST und NOLIST müssen jeweils innerhalb einer Programmeinheit stehen und gelten nur für diese.

Einige wichtige Übersetzer-Direktiven sollen hier vorgestellt werden. Man beachte auch die Erklärungen, die in Zusammenhang mit der Erläuterung von Optimierungs- und Vektorisierungsmöglichkeiten im Kapitel 12 gegeben werden.

In den folgenden Abschnitten sind mögliche Werte für *value* angegeben.

Beispiel:

```
CDIR$ NOLIST
CDIR$ NOCODE,IVDEP
```

8.2.1 Direktiven zur Steuerung der Ausgabeliste

Diese Direktiven wirken nur dann, wenn sie nicht im Widerspruch zu einer der beim CFT77-Aufruf angegebenen Optionen *L, m, h, g, s* oder *x* stehen (siehe 8.1.1).

EJECT	Beim Auflisten der Quelle wird eine neue Seite begonnen.
NOLIST	Unterdrückt die Ausgabeliste für den folgenden Teil der Programmquelle. Wirkt über die folgende END-Anweisung hinaus.
LIST	Schaltet die Erzeugung der Ausgabeliste wieder ein. Die Ausgabe erfolgt nach *stderr* oder, falls die Option *-l listfile* verwendet wurde, nach *listfile*.
NOCODE	Unterdrückt die Liste des generierten Codes vom Anfang des nächsten Optimierungsblocks an.
CODE	Erzeugt vom laufenden Optimierungsblock an die Liste des generierten Codes bis zu einer NOCODE-Direktive.

8.2.2 Direktiven zur Vektorisierung und Optimierung

Die Direktiven zur Vektorisierung setzen voraus, daß *-o full* (Standard) als Übersetzer-Option gewählt wurde (siehe 8.1.2).

NOVECTOR	Es wird eine Vektorisierung in den auf die Direktive folgenden Programmteilen verhindert.
VECTOR	Hebt NOVECTOR wieder auf, d.h., innere DO-Schleifen werden möglichst vektoriell abgearbeitet.
SUPPRESS	Diese Direktive unterdrückt die skalare Optimierung an der Stelle ihres Auftretens und verhindert die Vektorisierung der DO-Schleife, in der sie angegeben ist. Im Unterschied zu anderen Übersetzerdirektiven wirkt SUPPRESS nur dann, wenn die umgebende Codesequenz zur Laufzeit auch ausgeführt wird, d.h. ein SUPPRESS z.B. in einem IF-Zweig wirkt sich nur dann aus, wenn dieser Zweig auch durchlaufen wird.
IVDEP	*Ignore vector dependencies*: In der folgenden inneren Schleife werden Abhängigkeiten von Feldindizes, die einer Vektorisierung im Wege stehen können, nicht beachtet (siehe 12.4.4, Abhängigkeiten). Anmerkung: Die unbedachte Verwendung von IVDEP kann zu falschen Ergebnissen führen (siehe 12.4).
SHORTLOOP	Die Verwendung dieser Direktive kann zu einer kürzeren Ausführungszeit der nächsten inneren Schleife führen, falls diese Schleife vektorisiert wird. Sie darf nur dann verwendet werden, wenn bekannt ist, daß diese Schleife mindestens 1 und höchstens 64 mal durchlaufen wird. Andernfalls sind die Ergebnisse nicht definiert.
BL NOBL	*Bottom load*: Die Direktiven BL und NOBL ermöglichen bzw. verhindern, daß Operanden für den nächsten Durchlauf einer skalaren Schleife schon vor Abfrage des Endekriteriums geladen werden können (prefetching). Anmerkung: *prefetching* wird auch im letzten Schleifendurchlauf vorgenommen, wobei Zugriffe auf Feldelemente außerhalb der deklarierten Feldlänge vorkommen können. Da diese Feldelemente nicht weiterverarbeitet werden, ist das unproblematisch; es sei denn, die entsprechende Adresse liegt außerhalb des dem Benutzer zugeteilten Speicherbereiches; in diesem Fall kann ein *operand range error* auftreten.

8.2.3 Direktiven zur Überwachung und Fehlersuche

FLOW	Mit FLOW kann der FLOWTRACE (F) des Übersetzers für einzelne Unterprogramme eingeschaltet werden. Voraussetzung für Flowtrace ist das Ersetzen von CALL EXIT bzw. CALL ABORT durch STOP oder END. Diese Direktive ist sinnvoll für kleine, häufig gerufene Unterprogramme, um den zusätzlichen Rechenzeitbedarf der FLOWTRACE-Option zu begrenzen.
NOFLOW	Hebt FLOW wieder auf.
BOUNDS [a][,b] ...	Für alle oder die angegebenen Felder wird bei jedem Zugriff geprüft, ob die Feldgrenzen eingehalten werden (sehr rechenzeitaufwendig, keine Vektorisierung).

NOBOUNDS[a][,b] ... Hebt eine durch BOUNDS gesetzte Überprüfung für alle oder die angegebenen Felder wieder auf.

8.2.4 Sonstige Direktiven

INTEGER = n	Diese Direktive bestimmt die Länge von INTEGER-Zahlen für alle INTEGER-Variablen der jeweiligen Programmeinheit.
= 46	Anmerkung: Bei INTEGER = 46 wird ein Überlauf über 2**46-1 nicht erkannt.
= 64	Die Direktive muß direkt hinter einer PROGRAM-, FUNCTION- oder SUBROUTINE-Anweisung stehen.

8.3 Übersetzer-Optionen für CFT

In diesem und dem folgenden Kapitel 8.4 werden nur die Abweichungen von CFT gegenüber CFT77 beschrieben. Im Bereich der Optionen gibt es zum Teil erhebliche Unterschiede.

Ein Übersetzeraufruf mit dem FORTRAN-Programm in der Datei *quelle.f* kann bei Benutzung von Optionen so aussehen:

Beispiel:

```
cft -d p -e Afz -i 64 -o noifcon,fastmd quelle.f
```

8.3.1 Optionen zur Steuerung der Ein-/Ausgabe

Allgemeine Annahme: Das FORTRAN-Programm liegt in der Datei *quelle.f* .

quelle.f	Name der Datei, in der die Quelle steht; sollte mit *.f* enden.
-l listfile	*listfile* bezeichnet den Namen der Datei, in die die Übersetzerliste geschrieben wird; die Angabe *-eL* hat jedoch Vorrang (s.u.).
-b binfile	*binfile* bezeichnet den Namen der Datei, in die der Übersetzer die binären Lademodule
quelle.o	schreibt.
-c calfile	Es wird eine Datei erzeugt, in der Pseudo-Code für den CRAY-Assembler (CAL)
quelle.s	abgelegt wird. Der erzeugte Text kann nach wenigen Handkorrekturen als Eingabe für CAL dienen. DATA-Anweisungen werden nicht unterstützt. Die Datei wird nur erzeugt, wenn *-eC* gesetzt ist.
-v msgs	CFT klammert alle DO-Schleifen in der Ausgabeliste. Ferner werden Begründungen für die Nichtvektorisierung innerer DO-Schleifen ausgegeben. Die Option impliziert *-ed* .
nomsgs	Die begründenden Meldungen unterbleiben.
-A aids	Steuert die Anzahl der Meldungen, die ausgegeben werden, wenn bei einer inneren Schleife die Vektorisierung verhindert ist.
loopnone	Keine Meldungen.
looppart	Bis zu drei Meldungen je Compilationsblock, bis zu 100 Meldungen für eine Übersetzung.
loopall	Alle Meldungen.

-e onstring bzw. -d offstring:

Die in Kap. 8.1.1 angegebenen Optionen sind auch bei CFT vertreten mit folgenden Abweichungen: Die Option *-e/-d m* ist in *-v messages* enthalten (s.o.), es gilt die Voreinstellung *-e s* und die Option *-e/-d S* wird durch *-c calfile* formuliert (s.o.).
Ferner gibt es zusätzliche Möglichkeiten zur Ein-/Ausgabesteuerung:

-e/-d b	*Block*: Gibt die Zeilennummer und Speicheradresse jedes Compilationsblocks aus (sinnvoll in Verbindung mit einigen Optionen zur Fehlersuche; es läßt sich so eine Verbindung zwischen Zeilennummer und Speicheradresse herstellen).
-e/-d c	*COMMON*: Druckt Namen und Längen von COMMON-Blöcken.
-e/-d d	*DO*: Erzeugt eine Tabelle aller DO-Schleifen.
-e/-d l	*List directives*: Ermöglicht die Erkennung von Direktiven zur Steuerung der Übersetzerlisten (siehe 8.2.1).
-e/-d t	*Symbol table*: Gibt nach jeder Programmeinheit eine Symboltabelle aus (bei großen Programmen sehr umfangreich).
-e/-d y	Es werden die verwendeten Übersetzeroptionen und, je Programmeinheit, die Übersetzerdirektiven protokolliert.
-e/-dL	*Listing file*: Die FORTRAN-Quelle wird in die Datei *quelle.l* geschrieben. Die eventuell angegebene Option *-l listfile* (s.o.) wird ignoriert.

8.3.2 Optionen zur Vektorisierung und Optimierung

Von den in Kap. 8.1.2 angegebenen Optionen ist *-o full* mit den zugehörigen Alternativen bei CFT nicht verfügbar. Ferner fehlt die Option -I (*inlining*). Es gibt jedoch eine große Anzahl weiterer Optionen.

Mit den folgenden drei Optionswerten wird die Behandlung von IF-Anweisungen gesteuert (siehe 12.4 und 8.4.2: Direktiven NOIFCON und RESUMEIFCON):

-o noifcon partialifcon fullifcon	Bei *noifcon* werden IF-Anweisungen der Form IF (Bedingung) Zuweisung nur vektorisiert, falls der Übersetzer diese durch die Intrinsic-Funktionen MAX bzw. MIN ersetzen kann. Bei *partialifcon* vektorisiert CFT alle IF-Anweisungen dieser Form, sofern die Zuweisung nicht Divisionen oder Aufrufe von Standardfunktionen enthält. Der Übersetzer vektorisiert bei *fullifcon* auch solche IF-Anweisungen der obigen Form, die in der Zuweisung Divisionen oder Standardfunktionen enthalten.

Die beiden folgenden Optionswerte definieren die Genauigkeit und die Geschwindigkeit der INTEGER-Multiplikation und -Division:

-o fastmd	Der Übersetzer benutzt bei *fastmd* die schnelleren Operationen mit 46 Bit. Ein Überlauf wird hierbei nicht erkannt.
slowmd	Der Übersetzer führt bei *slowmd* alle INTEGER-Operationen mit der vollen 64-Bit Genauigkeit aus.

-o safedorep fulldorep nodorep	Einzeilige DO-Schleifen werden nur dann durch eine Routine aus SCILIB ersetzt, wenn keine Abhängigkeiten (siehe 12.4.2) und kein EQUIVALENCE vorliegen. Wenn *fulldorep* angegeben ist, werden beim Ersetzen dieser DO-Schleifen Abhängigkeiten und EQUIVALENCE nicht berücksichtigt (siehe gleichnamige Direktiven in 8.4.2). *nodorep* verhindert jede automatische Ersetzung durch Routinen aus SCILIB.
-o safeif unsafeif	Zur Optimierung können Befehle vertauscht werden. Bei *safeif* werden Befehle nicht vor Sprunganweisungen verschoben.
-o invmov partinvmov noinvmov	Mit diesen Optionen wird festgelegt, ob invarianter Code durch eine IF-Anweisung vor die Schleife gezogen werden soll. Durch *invmov* geschieht das ohne Einschränkung, durch *partinvmov* nur dann, wenn keine bedingten Divisionen und keine Aufrufe von FUNCTION-Unterprogrammen vorliegen; *noinvmov* verhindert jegliche Verlagerung von Code.
-o recurrence norecurrence	*recurrence* ermöglicht die Vektorisierung von Reduktionsschleifen (d.h. Schleifen, in denen ein Skalar aus einem vektorisierbaren Ausdruck berechnet wird); *norecurrence* unterdrückt die Vektorisierung solcher Schleifen.

-o vsearch vsearch64 novsearch	*vsearch* erlaubt die Vektorisierung von Suchschleifen (d.h. Schleifen, die durch einen bedingten oder unbedingten Sprung innerhalb eines bedingten Blockes verlassen werden können). Die interne Vektorlänge ist 8 bei *vsearch* und 64 bei *vsearch64*.
-o bl nobl	*Bottom Load*: In skalaren Schleifen können die nächsten Operanden schon geholt werden, bevor das Endekriterium abgefragt wird. Durch *nobl* wird "Bottom Load" und der dabei mögliche "Out of Range"-Fehler verhindert.
-o nobtreg btreg	*B- und T-Register*: Alle vom Benutzer definierten Variablen liegen im Speicher. B- und T-Register werden nur für Hilfsvariable und Zwischenergebnisse verwendet. Wenn *btreg* angegeben ist, können bis zu 24 lokale Variable eines Unterprogramms, die nicht in SAVE-, DATA-, COMMON- oder NAMELIST-Anweisungen vorkommen, in T-Registern abgelegt werden. Sie werden am Anfang nicht vorbesetzt und sind nach RETURN oder END undefiniert.
-o cvl nocvl	*Conditional Vector Loops*: Für Schleifen mit ungewissen Abhängigkeiten (siehe 12.4.2) wird vektorieller und skalarer Code erzeugt. Zur Laufzeit entscheidet sich, welche Version zu durchlaufen ist. *nocvl* legt fest, daß für Schleifen mit ungewissen Abhängigkeiten nur skalarer Code erzeugt wird.
-o keeptemp killtemp	Bei *keeptemp* haben Variable, die als temporäre Vektoren (siehe 12.4.2) verwendet werden, nach Durchlaufen der Vektorschleife den korrekten Wert. Bei *killtemp* sind sie nach Durchlaufen der Vektorschleife undefiniert.
-u unroll 3	Innere DO-Schleifen mit einer konstanten Zahl von bis zu *unroll* Durchläufen werden abgerollt (siehe 12.3.4). *unroll* ist maximal 9. *-u 0* verhindert das automatische Abrollen innerer Schleifen.
-M maxblock 4000	CFT kann Blöcke von Anweisungen mit einer Länge bis zu *maxblock* Worten optimieren und vektorisieren. Werte größer als die Voreinstellung 4000 können die Optimierung verbessern, aber auch zum Abbruch des Übersetzers führen. *-M 1* verhindert jede Optimierung oder Vektorisierung.

-e onstring bzw. -d offstring:

-e/-d v	*Vectorize*: Schaltet die Vektorisierung von DO-Schleifen ein bzw. aus. Das Ausschalten dieser Option kann in Einzelfällen sinnvoll sein, um den Vektorisierungsgrad eines Programms zu ermitteln.

8.3.3 Optionen zur Überwachung und Fehlersuche

Anmerkung zur Option *-m n* (message level) aus Kap. 8.1.3: Für *n* sind im Unterschied zu CFT77 die Werte 1,..,5 zulässig, wobei *n = 1,..,4* die gleichen Bedeutungen hat und *n = 5* bewirkt, daß Fehlermeldungen unterdrückt werden, soweit es sich nicht um fatale Fehler handelt.

-E errfile	*errfile* bezeichnet den Namen der Datei, in die Fehlermeldungen mit höherem Niveau als *-m n* ausgegeben werden; default ist *stderr*.

-e onstring bzw. -d offstring:

Alle in Kap. 8.1.3 beschriebenen Optionen sind auch für CFT gültig. Ferner gibt es zusätzlich die folgende Option:

-e/-d i	*Internal labels*: Gibt die vom Übersetzer generierten Anweisungsmarken in die Symboltabelle aus. Ähnlich wie *-e b* ist diese Option in Verbindung mit einigen Optionen zur Fehlersuche sinnvoll, da sie einen Bezug zwischen vom Übersetzer generierten Marken und den zugehörigen Adressen herstellt.

8.3.4 Optionen zur Programmportabilität

Zusätzlich zu den für CFT77 beschriebenen Optionen aus Kap. 8.1.4 gibt es für CFT noch die folgende Option:

-e/-d m — *Machine characteristics*: Gibt die Eigenschaften der eingestellten Zielmaschine aus.

8.3.5 Sonstige Optionen

Die in Kap. 8.1.5 beschriebenen Optionen gelten auch für CFT, wobei die Option *-i intlen* die Werte 64 (default) und 24 zuläßt und durch die Option -V statistische Angaben nur nach *stderr* geschrieben werden. Ferner sind zusätzlich folgende Optionen möglich:

-e onstring bzw. -d offstring:

-e/-d e — Mit dieser Option kann die Erkennung von Übersetzer-Direktiven (s.u.) gesteuert werden. Default: Direktiven werden erkannt.

-e/-d u — Bei *-d u* werden INTEGER*2-Variable in 64 Bits abgelegt, sonst in 24 Bits.

-e/-d w — Reelle, einfach genaue Gleitpunktoperationen können als Unterprogrammsprünge in externe Routinen ausgeführt werden.

-e/-d S — Wirkt wie eine SAVE-Anweisung mit leerer Liste, d.h. allen vom Benutzer definierten Variablen wird Speicher fest zugewiesen. *-o btreg* wird überlagert (siehe 8.3.2). Im Zusammenhang mit *-a stack* werden nur vom Übersetzer erzeugte Variable auf den *stack* gelegt.

8.4 Übersetzer-Direktiven für CFT

Die allgemeinen Angaben aus Kap. 8.2 gelten im wesentlichen auch für den CFT. Allerdings ist das Verhältnis zwischen Direktiven und Optionen im Kommando hier anders geregelt: Alle Übersetzer-Direktiven werden nur erkannt, wenn beim CFT-Aufruf *-e e* (default) eingestellt ist (siehe 8.3.5). Weitere Einschränkungen für Gruppen von Direktiven sind möglich (siehe unten).

8.4.1 Direktiven zur Steuerung der Ausgabeliste

Es gelten die in Kap. 8.2.1 beschriebenen Direktiven.

8.4.2 Direktiven zur Vektorisierung und Optimierung

Die in Kap. 8.2.2 beschriebenen Direktiven sind auch bei CFT möglich, wobei NOVECTOR eine zusätzliche Variante zuläßt:

NOVECTOR[=n] — Bei Angabe von *n*, einer ganzen Zahl zwischen 0 und 64, bezieht sich die Verhinderung der Vektorisierung nur auf Schleifen, deren Wiederholungszahl *I* (iteration count) für den Übersetzer erkennbar eine Konstante ist und für die I <= n gilt.

Im Bereich der Vektorisierung und Optimierung gibt es für CFT darüber hinaus eine große Anzahl weiterer Direktiven:

NORECURRENCE [=n] =2

Diese Direktive legt fest, ob Schleifen, die eine akkumulierende Zuweisung enthalten (*reduction*, Sonderfall einer *recurrence relation*), skalar oder vektoriell abgearbeitet werden sollen.
Akkumulierende Zuweisungen der Form

$S = S + e$ oder $S = S * e$

kann CFT vektorisieren, wenn S eine skalare Variable und e ein vektorisierbarer Ausdruck ist.
Ohne die Angabe von n wird eine Vektorisierung aller folgenden akkumulierenden Schleifen verhindert. Bei Angabe von n werden nur die akkumulierenden Schleifen skalar abgearbeitet, deren Wiederholungszahl I (iteration count) für den Übersetzer erkennbar eine Konstante ist, für die I < = n gilt.

NEXTSCALAR — Mit dieser Direktive wird die Vektorisierung für die folgende DO-Schleife abgeschaltet.

VSEARCH — Vektorisiert Suchschleifen mit der internen Vektorlänge 8.

VSEARCH64 — Vektorisiert Suchschleifen mit der internen Vektorlänge 64.

NOVSEARCH — Verhindert die Vektorisierung von Suchschleifen.

FASTMD
SLOWMD

Diese Direktiven entscheiden über INTEGER-Multiplikation und -Division im laufenden Optimierungsblock (siehe 8.3.2 Optionen *-o fastmd* und *-o slowmd*).
Bei FASTMD haben die Ergebnisse nur eine Genauigkeit von 46 Bit. Überlauf wird nicht festgestellt.
Bei SLOWMD wird die langsamere Ganzwortarithmetik (64 Bit) verwendet.

UNSAFEIF
SAFEIF

Die Direktiven UNSAFEIF und SAFEIF ermöglichen bzw. verhindern, daß in einem Block einzelne Befehle (außer Speichern und Division) über den Sprung einer IF-Anweisung hinaus verschoben werden können.
Beide Direktiven überlagern nur für den Block, in dem sie stehen, die Einstellung beim CFT-Aufruf (siehe 8.3.2).

ALIGN — Die Direktive ALIGN bestimmt, daß die folgende Anweisung an den Anfang des Befehlspuffers gelegt wird, wenn diese eine SUBROUTINE-, PROGRAM-, FUNCTION-, ENTRY- oder DO-Anweisung oder ein Sprungziel ist.

NOIFCON
RESUMEIFCON
PARTIALIFCON
FULLIFCON

IF conditions: Mit der Direktive NOIFCON kann die Optimierung von Anweisungen der Form

IF (Bedingung) Zuweisung,

sofern sie der Übersetzer nicht in Anweisungen mit den Intrinsic-Funktionen MAX bzw. MIN umwandeln kann, abgeschaltet und mit RESUMEIFCON auf die beim CFT-Aufruf angegebene Option zurückgeschaltet werden. PARTIALIFCON bzw. FULLIFCON wirken wie *-o partialifcon* bzw. *-o fullifcon* (siehe 8.3.2).

NODOREP
RESUMEDOREP
SAFEDOREP
FULLDOREP

DO loop replacement: Die Direktive NODOREP verhindert die Ersetzung einzeiliger DO-Schleifen durch eine Routine aus SCILIB. RESUMEDOREP kehrt zu der Einstellung beim CFT-Aufruf zurück. SAFEDOREP bzw. FULLDOREP wirken wie *-o safedorep* bzw. *-o fulldorep*.

NOCVL
CVL

Conditional vector loops: Mit diesen beiden Direktiven können die gleichnamigen Optionen beim CFT-Aufruf (siehe 8.3.2) überschrieben werden.
CVL bedeutet, daß für Schleifen mit ungewissen Abhängigkeiten (siehe 12.4.2) vektorieller und skalarer Code erzeugt wird. Zur Laufzeit entscheidet sich, welche Version zu durchlaufen ist.
Nach NOCVL wird für Schleifen mit ungewissen Abhängigkeiten nur skalarer Code erzeugt.

ROLL
UNROLL

Nach ROLL werden keine DO-Schleifen abgerollt. Nach UNROLL können innere DO-Schleifen mit einer konstanten Zahl von Durchläufen gemäß der Angabe beim CFT-Aufruf abgerollt werden (siehe 8.3.2 Option *-u unroll*).

INVMOV
PARTINVMOV
NOINVMOV

Mit diesen Optionen wird festgelegt, ob invarianter Code durch eine IF-Anweisung vor die Schleife gezogen werden soll. Durch INVMOV geschieht das ohne Einschränkung, durch PARTINVMOV nur dann, wenn keine bedingten Divisionen und keine Aufrufe von FUNCTION-Unterprogrammen vorliegen; NOINVMOV verhindert jegliche Verlagerung von Code.

8.4.3 Direktiven zur Überwachung und Fehlersuche

Zusätzlich zu den Direktiven zur Überwachung der Feldgrenzen (siehe 8.2.3) gibt es bei CFT die folgenden Schreibweisen:

BOUNDS (a,b,c)	Einschaltung der Bereichsüberprüfung für definierte Felder.
BOUNDS()	Ausschaltung der Bereichsüberprüfung (identisch mit NOBOUNDS).

8.4.4 Sonstige Direktiven

INT24v[,v,...] INT64v[,v,...]	Diese Direktiven bestimmen die Länge von INTEGER-Zahlen in Bits entweder für alle vereinbarten Variablen, die mit den Buchstaben I bis N anfangen, oder für die angegebenen Variablen *v*,... (siehe 8.3.5, Option *-i*). Anmerkung: Bei INT24 wird ein Überlauf über 2**23-1 nicht erkannt.

8.5 Der Präprozessor *fpp* und der Multitasking-Übersetzer *fmp*

Zusätzlich zu den FORTRAN77-Übersetzern CFT77 und CFT gibt es zwei weitere Übersetzer, den FORTRAN-Präprozessor *fpp* und den FORTRAN-Multitasking-Übersetzer *fmp*. Diese Übersetzer sind mit dem CFT77 bzw. CFT sowie anderen Werkzeugen zusammengefaßt zu den sogenannten Autotasking-Systemen *cf77* bzw. *cf*, die die Erzeugung von vektorisiertem und parallelisiertem Code ermöglichen. Für die Mehrheit der Benutzer ist bei der jetzigen Systemkonfiguration nur die Vektorisierung und weniger die Parallelisierung von Interesse. Hinzu kommt, daß in manchen Fällen das komplexe Autotasking-System schwierig zu bedienen ist. Es empfiehlt sich daher, neben den Übersetzern CFT77 und CFT den Präprozessor *fpp* separat zu verwenden.

Der Präprozessor *fpp* übersetzt von FORTRAN nach FORTRAN und analysiert dabei die Schleifen. Vektorisierbare Schleifen werden ebenso wie eindeutig nicht vektorisierbare Schleifen durch eine entsprechende Compiler-Direktive gekennzeichnet. Ist die Parallelisierung nicht ausgeschaltet, so werden auch Vorbereitungen für eine Parallelisierung von DO-Schleifen getroffen: die von *fpp* gefundene Parallelität wird in Form von Autotasking-Direktiven im Quellcode niedergelegt. Dabei hat die Vektorisierung jedoch Vorrang vor der Parallelisierung. Der einfachste Aufruf des *fpp* mit abgeschalteter Parallelisierung (-dc) lautet:

```
fpp -dc quelle0.f > quelle.f
```

Es gibt eine Vielzahl von Möglichkeiten, den Übersetzungsvorgang durch Optionen und Direktiven zu steuern, die hier aber nicht näher beschrieben werden. Eine ausführliche Darstellung der Steuerung und der Leistungen des *fpp* findet sich im *ZIB Technical Report TR 90-4* (siehe Kapitel B.3) und in DOC,CRYFORT,FPP.

9. PASCAL Übersetzer-Optionen und -Direktiven

Von der CRAY Research Inc. wird für die CRAY X-MP z.Zt. die PASCAL-Version 4.0 angeboten, die bis auf drei Einschränkungen dem Standard ISO/DIS 7185 der International Standards Organization (ISO) entspricht.

Ausführliche Informationen zur Benutzung von PASCAL an der CRAY, über den Sprachumfang sowie über die CRAY-spezifischen Einschränkungen und Erweiterungen findet man in dem Handbuch CRAY COMPUTER SYSTEMS, PASCAL REFERENCE MANUAL, SR-0060 E.

Mit dem Übersetzer kann der PASCAL-Quellcode in Maschinencode für die Rechner CRAY Y-MP, CRAY-2, CRAY X-MP und CRAY-1 übersetzt werden. Im folgenden soll anhand eines einfachen Beispiels die Benutzung von PASCAL an der CRAY X-MP des ZIB erläutert werden.

9.1 Ein einfaches PASCAL-Beispiel

Übersetzen, Binden, Laden und Ausführen eines Programms

```
#USER=userid PW=password
#QSUB -lm 1Mw
#QSUB -lT 10
cat << '@@@' > prog.p
PROGRAM DATEIEN (INPUT,OUTPUT);
CONST SIZE = 5;
VAR FELD : ARRAY (.1..SIZE.) OF INTEGER;
      I : INTEGER;
BEGIN
      FOR I := 1 TO SIZE DO
            READLN(FELD(.I.));
      FOR I := SIZE DOWNTO 1 DO
            WRITELN(FELD(.I.));
END.
@@@
cat << '@@@' > daten
1
2
3
4
5
@@@
pascal -i prog.p
segldr a.o
a.out < daten
```

Der oben aufgeführte Job stellt die einfachste Möglichkeit dar, ein PASCAL-Programm zu übersetzen, zu binden, zu laden und auszuführen.

Die wesentlichen UNICOS-Kommandos für PASCAL:

pascal -i prog.p	Hiermit wird der PASCAL-Übersetzer aufgerufen. Das übersetzte Programm liegt anschließend in der Datei *a.o* (Voreinstellung).

segldr a.o — Mit dieser Anweisung wird der Lader aufgerufen. Durch den Lader werden automatisch die benötigten Routinen aus der PASCAL-Laufzeitbibliothek (PSCLIB, enthalten in der Datei */lib/libp.a*) mit dem übersetzten PASCAL-Programm zusammengebunden. Es können für den Lader weitere Parameter angegeben werden. (siehe Kap. 4.3)

a.out < daten — Start des gebundenen PASCAL-Programms unter Einbeziehung der Testdaten in der Datei *daten*.

9.2 Aufruf des PASCAL-Übersetzers

Die vollständige PASCAL-Anweisung hat folgendes Format:

pascal -i idn -l ldn -b bdn -o list -V -C cpu

Die Voreinstellung (Default) ist unterstrichen; In Klammern gesetzte Parameter sind optional.

-i idn
stdin — *idn:* Name der Datei, die den PASCAL-Quellcode enthält.

-l ldn
stdout — *ldn:* Name der Datei für die Aufnahme der Meldungen des Übersetzers. Durch die Angabe *-l 0* werden die Meldungen mit Ausnahme der *fatal error messages* unterdrückt..

-b bdn
a.o — *bdn:* Name der Datei für die Aufnahme des erzeugten Objektmoduls.

-o list — *list:* Durch Kommata getrennte Auflistung von Optionen (directives) zur Steuerung der Übersetzung. Einige Optionen können auch im PASCAL-Programm innerhalb von Kommentaren stehen. Einige Optionen dürfen nur beim Aufruf des Übersetzers oder nur im PASCAL-Programm innerhalb von Kommentaren verwendet werden (weitere Erläuterungen siehe Kapitel 9.3).

Voreinstellung:
A+, AL-, BP-, BREG=8, BT-, C-, Debug-, DM0, E+, FE-, G-, L-, ML3, O+, P-, P24, R+, RV-, ST-, T+, TREG=8, U-, V+, W-, X-, Z+

-V — Ausgabe der Compiler-Versionsnummer und anderer Statistik (Übersetzungszeit, Speicherbedarf, Größe des Binärprogramms) in das Error-File *stderr*.

-C cpu — *cpu* hat das Format: [cputyp][,cpuchar1][,cpuchar2] ... [,cpucharn];
cputyp ist der Rechnertyp, für den der Code erzeugt werden soll;
cpuchar kennzeichnet eine Hardware-Eigenschaft der Zielmaschine.

Voreinstellung für *cpu* (nur mit den wichtigsten Hardware-Eigenschaften):
cray-xmp,noema,nocigs,vpop

9.3 Steuerung des PASCAL-Übersetzers

Der Übersetzungsvorgang wird mit globaler Wirkung durch die Parameter der Option *-o* und mit lokaler Wirkung durch die Übersetzer-Direktiven im PASCAL-Quellcode gesteuert. Im Gegensatz zum PASCAL Reference Manual SR-0060 der Firma CRAY Research werden hier nur die im Quellcode auftretenden Steuerungsparameter als Übersetzer-Direktiven bezeichnet.

Es gibt drei Typen von Steuerungsparametern: solche, die nur in der Option *-o* verwendet werden dürfen; solche, die nur im Quellcode als Übersetzer-Direktiven auftreten dürfen und solche, die an beiden Stellen erlaubt sind. Ausschließlich in der Option *-o* des Übersetzers können die Parameter BREG, C, DEBUG, DM, E, FE, ISO, ML, O, P24, P32, TREG, V, W, X, XI, XP, XV und Z angegeben werden; nur als Direktive im Quellcode dürfen F, I, R^, R*, VI und VN auftreten.

Die Wirkung der Übersetzer-Direktiven erstreckt sich im allgemeinen auf die nachfolgende Programmeinheit im Quelltext. Das Format der Übersetzer-Direktiven im PASCAL-Programm ist

```
(*#param1:param2: ... paramn*)
```

Die meisten Parameter können durch nachfolgende + oder - -Zeichen ein- bzw. ausgeschaltet werden.

Beispiel:

Angabe von Parametern in der Option *-o*:

```
pascal -oBT+,R-,V+,ISO1  -i prog.p
```

Angabe von Parametern als Übersetzer-Direktive im PASCAL-Programm:

```
(*#BT+:R-:V+:ISO1*)
```

9.3.1 Parameter zur Steuerung der Ein- und Ausgabe

C+, C-
: *C+* schreibt den vom Übersetzer erzeugten Assembler-Code (Pseudo-CAL) auf *stdout* oder auf die Übersetzer-Liste, d.h. auf die mit der Option *-l* angegebene Datei.

F
: *F* erzwingt als Übersetzer-Direktive einen Seitenvorschub in der Übersetzerliste.

I file
: Der Inhalt der Datei *file* wird im Quellprogramm an der Stelle dieser Übersetzer-Direktive eingefügt. Rekursives Einfügen ist nicht möglich.

L+, L-
: *L+* schreibt das Quellprogramm in die Übersetzerliste (impliziert ML0); L- unterdrückt dies (wird impliziert von der Übersetzer-Option *-l0*).

MLn
: *Message level: n* gibt das höchste Niveau der Übersetzermeldungen an, die unterdrückt werden sollen:
  - ML0: alle Meldungen werden ausgegeben.
  - ML1: *Comment:* Optimierungsinformationen einschließlich der Meldungen über nicht vektorisierbare FOR-Schleifen werden unterdrückt.
  - ML2: *Note:* Kommentare zur Effizienz des Programms werden zusätzlich unterdrückt.
  - ML3: *Caution:* Anmerkungen über mögliche Fehler werden zusätzlich unterdrückt.
  - ML4: *Warning:* Anmerkungen über wahrscheinliche Fehler werden zusätzlich unterdrückt; Meldungen zu Fehlern des Niveaus 5 können nicht unterdrückt werden.

P+, P-
: *P+* erzwingt einen Seitenvorschub in der Übersetzerliste vor jeder Routine.

ST=string, ST-
: Die ersten 72 Zeichen von *string* werden als Untertitel in der zweiten Zeile der Seitenüberschrift angeführt. Sonderzeichen sind durch einen vorangestellten \ zu kennzeichnen.

Un, U+, U-	*n* gibt die Anzahl signifikanter Spalten im Quellprogramm an; U+ entspricht 72, U- 120 signifikanten Spalten (1 < = n < = 140).
X+, X-	Durch *X+* (entspricht XP+ und XV+) wird eine globale Querverweis-Tabelle erstellt; bei *X-* (entspricht XP- und XV-) wird keine erstellt.
XI+, XI-	*XI+* fügt der Übersetzerliste Informationen über *identifier* bei.
XP+, XP-	*XP+* erzeugt Querverweise auf alle Prozeduren.
XV+, XV-	*XV+* erzeugt Querverweise global für alle verwendeten Namen.

9.3.2 Parameter zur Steuerung der Vektorisierung und Optimierung

AL-, AL+	*AL+* legt den Anfang von Prozeduren und Funktionen an den Anfang des Befehlspuffers. Dies vergrößert zwar den Code, bringt in Einzelfällen aber Laufzeitvorteile.
BREG=n, BREG=8, BREG-	Durch *BREG=n* (0< =n< =64) werden die ersten n mit VAR deklarierten 24-Bit-Variablen (z.B. Zeiger-, I24- und CHAR-Variablen) eines Programmoduls in B-Registern gehalten. Bei *BREG-* werden keine B-Register für Benutzer-Variablen verwendet.
E+, E-	*E+* fügt kleine Prozeduren und Funktionen in die aufrufenden Programmeinheiten ein (*inline expansion*); äquivalent dazu ist eine INLINE-Anweisung in den Deklarationen aller Prozeduren und Funktionen.
FE+, FE-	Bei *FE+* werden Boolsche Ausdrücke vollständig ausgewertet; z.B. wird im Ausdruck *bval AND f(x)* die Funktion *f* auch bei *bval=FALSE* aufgerufen. Das kann erforderlich sein, wenn *f* Seiteneffekte hat. Bei *FE-* wird nur die minimale Anzahl von Teilausdrücken zur Feststellung des logischen Wertes ausgewertet.
I64, I46	Gibt die Genauigkeit an, mit der Integer-Operationen durchgeführt werden; 46-Bit-Operationen sind schneller als 64-Bit-Operationen, aber bei 46 Bit wird kein Überlauf erkannt.
O+, O-	*O+* schaltet die skalare Optimierung ein.
TREG=n, TREG=8, TREG-	Mit *TREG=n* (0< =n< =64) werden die ersten *n* mit VAR deklarierten 64-Bit-Variablen (z.B. I64-, REAL- und BOOLEAN-Variablen) eines Programmoduls in T-Registern gehalten. Bei *TREG-* werden keine T-Register für Benutzer-Variablen verwendet.
V+, V-	*V+* schaltet den Autovektorisierer für FOR-Schleifen ein; zur Vektorisierung muß mit *RA-* die Laufzeitüberprüfung der Feldgrenzen abgeschaltet sein.
VI	Vektorabhängigkeiten in der nächsten FOR-Schleife werden beim Vektorisieren ignoriert.
VN	Die nächste FOR-Schleife wird nicht vektorisiert.

9.3.3 Parameter zur Laufzeitüberwachung und Fehlersuche

BP+, BP-

BP+ ermöglicht das Setzen von *breakpoints* beim Schreiben in Speicherzellen; durch Aufruf einer Bibliotheksroutine wird jedesmal eine Meldung ausgegeben, wenn sich der Wert der durch den Aufruf

```
P$BREAK(K,TRUE)
```

angegebenen Speicherzelle K ändert (K ist eine statische Variable). Die Routine P$BREAK ist durch

```
PROCEDURE P$BREAK(VAR location:INTEGER;isvariable:BOOLEAN);
IMPORTED;
```

im Hauptprogramm zu deklarieren.

BT+, BT-

BT+ schaltet für das gesamte Programm die Messung des Zeitverbrauchs, aufgeschlüsselt nach den einzelnen Programmodulen, ein (.i.Flowtrace;). Soll ein Flowtrace nur für einzelne Prozeduren und Funktionen aktiviert werden, so sind die Routinen FLOWENTR und FLOWEXIT zu importieren und am Anfang bzw. Ende der jeweiligen Routinen aufzurufen; weiterhin ist FLOWSTOP zu importieren und am Ende des Programms aufzurufen, sowie *BT* auszuschalten (Importieren ist eine CRAY-Erweiterung!).

DEBUG-, DEBUG+

Mit *DEBUG+* erhält die vordefinierte, im Programm verwendbare Boolsche Konstante $DEBUG den Wert TRUE, mit *DEBUG-* den Wert FALSE.

DMn

debug-mode: legt fest, welche Code-Adressen in der Debug-Symboltabelle vermerkt werden und welche Optimierungen vorgenommen werden. Bei n>1 werden skalare Optimierung und automatische Vektorisierung abgeschaltet.

- DM0 — Es wird keine Debug-Symbol-Tabelle erstellt.
- DM1 — Die Debug-Symbol-Tabelle enthält Informationen über Typen, Variablen und Konstanten, die Verwendung nichtlokaler Variablen in Prozeduren und die Adressen von Benutzer-definierten Marken und Prozedur-Eintrittspunkten.
- DM2 — Die Debug-Symbol-Tabelle enthält gegenüber DM1 zusätzlich die Adressen der vom Übersetzer erzeugten Marken in strukturierten Anweisungen.
- DM3 — In der Debug-Symbol-Tabelle werden zusätzlich die Adressen jeder Anweisung aufgeführt.

G+, G-

G+ besetzt Kellervariablen mit nicht definierten (negativen) Werten vor - hilfreich zum Auffinden von nicht-initialisierten Variablen.

R+, R-

R+ erzeugt zusätzlichen Code für Laufzeitüberprüfungen (identisch mit RA+, RR+, RP+, RS+, RL+, RN+); R- unterdrückt dies (entspricht RA-, RR-, RP-, RS-, RL-, RN-).

RA+, RA-

RA+ erzeugt zusätzlichen Code zur Laufzeitüberprüfung von Feldgrenzen.

RL+, RL-

RL+ erzeugt zusätzlichen Code zur Laufzeitüberprüfung der Kompatibilität von Feldern; kompatibel sind Felder mit gleicher Anzahl von Dimensionen und gleicher Größe in den einzelnen Dimensionen.

RN+, RN-

RN+ erzeugt zusätzlichen Code zur Laufzeitüberprüfung der Anzahl der an externe PASCAL-Prozeduren übergebenen Parameter

RP+, RP-
RPN — *RP+* erzeugt zusätzlichen Code zur Laufzeitüberprüfung von Speicherzugriffen mittels Zeigervariablen; mit *RPN* werden Zeiger nur auf NIL-Werte geprüft.

RR+, RR- — *RR+* erzeugt zusätzlichen Code zur Laufzeitüberprüfung von Wertzuweisungen an Variablen mit einem Teilbereichstyp (subrange type).

RS+, RS- — *RS+* erzeugt zusätzlichen Code zur Laufzeitüberprüfung von Mengenoperationen.

RV+, RV- — *RV+* erzeugt zusätzlichen Code zur Laufzeitüberprüfung in Strukturen vom Typ Datenverbund mit Varianten (variant records).

R^ — Sichert den momentanen Zustand der R-Optionen.

R* — Stellt den vorher mit R^ abgespeicherten Zustand der R-Optionen wieder her.

T+, T- — Mit *T+* wird zusätzlicher Code erzeugt zur Laufzeitüberprüfung von Kellergrenzen (stack overflow). Bei *T-* findet keine Überprüfung statt; fließt der Keller über, so kann die Halde (heap) überschrieben werden.

W+, W- — *W+* bewirkt, daß bei einem Abbruch des Programms *SYMDEBUG* aufgerufen wird. Dieses liefert ein *Traceback* und druckt die Werte aller in der Debug-Symbol-Tabelle aufgeführten Variablen aus (s.o., Parameter DM).

9.3.4 Sonstige Parameter

A+, A- — Der an die Shell zurückgelieferte Fehlerstatus beim Auftreten von Übersetzungsfehlern ist bei *A+* ungleich Null (Fehlerstatus), bei *A-* gleich Null.

ISO,
ISO0,
ISO1 — Wird der Parameter nicht angegeben (Voreinstellung), so werden über den ISO-Level 1 hinausgehende CRAY-spezifische Erweiterungen akzeptiert. Bei Angabe von ISO (identisch mit ISO1) wird ISO-Level 1 akzeptiert, bei ISO0 nur Level 0 (Unterschied: bei Level 1 werden auch *conformant arrays* akzeptiert)

P24, P32 — Benutzung von Zeigern (default 24 Bit) in gepackten Strukturen.

Z+, Z- — Erzeugen von wiedereintrittsfähigem (reentrant) Code.

9.4 Vektorisierung von PASCAL-Programmen

Um die Leistung der CRAY optimal zu nutzen, sollte ein PASCAL-Programm an die Vektoreigenschaften der CRAY angepaßt werden. Einzelheiten für die Ausnutzung der Vektoreigenschaften der CRAY findet man im PASCAL REFERENCE MANUAL SR-0060 (siehe Kapitel B.1). An dieser Stelle sollen nur einige grundsätzliche Informationen gegeben werden, die es einem Benutzer ermöglichen, die Vektoreigenschaften der CRAY zu nutzen, ohne das PASCAL-Programm völlig überarbeiten zu müssen.

Die Operanden konventioneller Rechneroperationen sind im allgemeinen einzelne Werte (Skalare). Die Operanden von Vektoroperationen sind dagegen Vektoren.

Vektor: — Zusammengehörige Folge von Skalaren, die mit konstantem Inkrement im Speicher abgelegt sind. In PASCAL: eindimensionale Felder, Spalten oder Zeilen einer Matrix.

Vektoroperation: — Arithmetische oder logische Operation, die alle Elemente eines Vektors "quasi gleichzeitig" verarbeitet.

In PASCAL werden CRAY-Vektoroperationen stets durch FOR-Schleifen, spezielle Anweisungen für die Feldverarbeitung (array processing constructs) oder durch den Aufruf von CRAY-spezifischen Unterprogrammen erzeugt. Minimale Voraussetzungen für die Erzeugung von Vektoroperationen sind:

* Benutzung von FOR-Schleifen, CRAY-spezifischen Unterprogrammen oder speziellen Anweisungen für die Feldverarbeitung (array processing constructs).
* Verwendung von eindimensionalen Feldern, Spalten oder Zeilen einer Matrix innerhalb von FOR-Schleifen (packed arrays werden nicht vektorisiert);
* die Option V+ muß gesetzt sein (Voreinstellung);
* es muß mindestens die Option RA- gesetzt werden (keine Voreinstellung!). Es kann notwendig sein, daß weitere Laufzeitüberprüfungen abgeschaltet werden müssen. Wenn man längeres Testen vermeiden möchte, empfiehlt es sich, die Option R- zu setzen, mit der diverse Laufzeitüberprüfungen abgeschaltet werden (siehe 9.3.3).

Der Übersetzer gibt dem Benutzer an, welche FOR-Schleifen vektorisiert werden konnten.

FORTRAN-Unterprogramme, die im UNICOS Math and Scientific Library Reference Manual (SR-2081 5.0) beschrieben werden, können auch aus PASCAL-Programmen aufgerufen werden. Es können in manchen Fällen Probleme entstehen, da PASCAL und FORTRAN nicht in allen Punkten kompatibel sind (siehe PASCAL REFERENCE MANUAL, FORTRAN-Direktive).

Beispiel für eine Vektorisierung eines PASCAL-Programms

```
#USER=userid PW=password
#QSUB -lm 1Mw
#QSUB -lT 10
set -ex
cat << '@@@' > prog.p
PROGRAM SIEVE (OUTPUT);
(* VGL. BSP. 8 AUS DEM CRAY PASCAL 4.0 REFERENCE MANUAL, S. 13-12 *)
CONST SIZE=500000;
TYPE  FLAGARRAY= ARRAY(.2..SIZE.) OF BOOLEAN;
VAR   FLAGS: FLAGARRAY;
      I,J : INTEGER;
BEGIN
      FOR I := 2 TO SIZE DO
            FLAGS(.I.) := TRUE;
      FOR I:=2 TO TRUNC(SQRT(SIZE)) DO
            IF FLAGS(.I.)
                  THEN FOR J := I+I TO SIZE BY I DO
                              FLAGS(.J.) := FALSE;
      J:=0;
      FOR I := 2 TO SIZE DO
            J := J+ORD (FLAGS(.I.));
      WRITELN (' END VECTORIZED PASCAL SIEVE; ', J:0,
            'PRIMES FOUND.')
END.
@@@
pascal -i prog.p -o RA-,RR-
segldr a.o
a.out
```

Das Programm SIEVE enthält vier FOR-Schleifen, von denen drei vektorisierbar sind. Damit die FOR-Schleifen tatsächlich vektorisiert werden können, müssen die Optionen RA- und RR- angegeben werden.

10. C Übersetzer-Optionen und -Direktiven

Eine kurzgefaßte Einführung mit Literaturhinweisen gibt es in Kapitel 4.2.3. Dort wird bereits erwähnt, daß es zwei verschiedene C-Compiler gibt: Den bisher gewohnten auf der Basis des Portable C-Compilers von AT&T, genannt PCC, sowie einen neuen, dem zukünftigen ANSI-Standard entsprechenden, genannt SCC. Die folgenden Ausführungen gelten für beide; Abweichungen werden beschrieben oder für SCC in Klammern dargestellt [...].

Der Aufruf des C-Compilers

Der Aufruf des C-Compilers hat gemäß SR-2024 [SR-2074] im Prinzip den folgenden Aufbau:

cc options file.c [scc options file.c]

Dabei sind:

cc [scc]	Aufruf des C-Compilers mit Präprozessor CPP und Lader.
options	Liste der zugehörigen Optionen.
file	C-Quelldatei; die zwei letzten Zeichen des Namens müssen *.c* sein (Suffix).

Der einfachste Aufruf zum Übersetzen, Binden und Starten eines C-Programms ist:

```
cc file.c                    [scc file.c]
```

Hier werden für die erforderlichen Optionen interne Voreinstellungen wirksam, so daß keine Angabe erfolgen muß. Für aufwendige Programmierungsarbeiten kann es zweckmäßig sein, in den Ablauf des in *cc* [scc] enthaltenen Mechanismus einzugreifen. Mit Hilfe von Optionen kann nach jedem der Schritte

Präprozessor, Compiler, Optimierung, Assemblierung und Binden

angehalten oder beliebig aufgesetzt werden. Die Zwischenergebnisse werden in Dateien abgelegt, deren Namen mit dem zum jeweiligen Schritt gehörigen Suffix versehen werden.

Andere Optionen ermöglichen spezielle Anweisungen an Präprozessor und Compiler oder die Vergabe eigener Dateinamen anstelle der voreingestellten.

Im folgenden werden einige Optionen stichwortartig aufgeführt, die z.B. für Testzwecke nützlich sein können:

-E, -P	nur Präprozessorlauf (verschiedene Ausgaben).
-# -## -###	keine Ausführung, erstellt nur Ablaufinformation.
-D, -U	wirken auf den *define*-Mechanismus des Präprozessors.
-c	das Binden wird unterdrückt (SEGLDR).
-o filename	für die Ausgabe-Datei wird ein Name vereinbart (anstelle von *a.out*).
-F	ermöglicht *Flowtrace*.
-g	richtet eine für *Debugging* erforderliche Tabelle ein.
-h options	beeinflußt die Code-Generierung, z.B. Abbruchkriterien, Speichernutzung, Vektorisierung und Protokollgestaltung.

Alle Optionen werden ausführlich beschrieben in SR-2024 [SR-2074] (C-Beschreibungen) sowie in SR-2011 (Kommandohandbuch), das für Präprozessor CPP und Lader SEGLDR noch zusätzliche Optionen anbietet.

Die Standard-Ein-/Ausgabedateien für die Kommandos *cc* und *scc* werden durch Suffixe zum Dateinamen gekennzeichnet:

.c bezeichnet die Quelldateien,
.o bezeichnet Objekt-Dateien (für den Lader),
.s bezeichnet Assembler-Dateien,
.i bezeichnet die Ausgabe des Präprozessors für den Compiler.

a.out ist der Standardname für die Ausgabedatei, in der das ablauffähige Programm steht.

Optimierung

Damit zur Verkürzung der Laufzeit der Übersetzer das Programm möglichst weitgehend vektorisieren kann, lohnt es sich, insbesondere diejenigen Programmteile, die viel Laufzeit benötigen, durch Programmabänderungen für die Vektorisierung günstig zu gestalten. Zum Auffinden der Passagen eines C-Programms, bei denen eine Abänderung lohnt, dienen die Optionen

-h vreport und -F (Flowtrace)

Das Problem der Abhängigkeiten (dependencies), die einer Vektorisierung entgegenstehen, wird zwar erkannt, kann aber ebenfalls nur vom Programmierer gelöst werden.

Der Standard C-Compiler SCC kann über *for*-Schleifen hinaus auch Schleifen mit Ausdrücken wie *while*, *gather/scatter*, *if* sowie mathematische Funktionen vektorisieren. Da dem SCC zur Optimierung ganz allgemein neue und effektivere Strategien zur Verfügung stehen, werden viele Programme schneller ablaufen als unter PCC.

Struktur von C-Programmen

Ein C-Programm besteht fast ausschließlich aus einer Reihe von "Funktionen" genannten Programmeinheiten, die sich gegenseitig oder auch selbst aufrufen. Es muß mindestens eine Funktion geben; diese muß MAIN heißen und ist nicht rekursiv. Diese Gliederung in Funktionen ist der Organisation in Haupt- und Unterprogrammen bei anderen Sprachen vergleichbar. So sind auch die verwendeten Namen meist nur innerhalb einer Funktion gültig.

Allgemeiner Aufbau:

funktionsname (parameterliste)
parameterdeklarationen -optional-
{inhalt, d.h. formulierte anweisungen}

Der Funktionsinhalt ist in geschweifte Klammern "{ }" eingeschlossen,
die einzelnen Anweisungen sind mit einem Semikolon ";" abgeschlossen.

Die Funktionen stehen über Parameterwerte, ihre jeweils eigenen Werte, globale Variable und Felder sowie ggf. gemeinsam genutzte Dateien untereinander in Beziehung. Die Parameter werden nach dem Verfahren *call by value* übergeben. Außerhalb von Funktionen treten nur spezielle Definitionen und Anweisungen an den Präprozessor auf.

Der Präprozessor ist eine Spezialität von C. Er bearbeitet die Programmquelle zuerst und interpretiert die für ihn bestimmten Anweisungen (sie beginnen mit dem Nummernkreuz "#"). Der Compiler bearbeitet erst danach die so veränderte Quelle.

Bei der Formulierung von C-Programmen wird eine große Zahl von Sonderzeichen verwendet, sowie die Groß- und Kleinschreibung der Buchstaben. Die Groß-/Kleinschreibung ist relevant; reservierte Worte in C müssen klein geschrieben werden.

Im SCC gibt es eine besondere Möglichkeit, ASCII-Zeichen darzustellen, die auf europäischen Tastaturen nicht vorhanden ist: Die Ersatzdarstellung mittels *trigraphs*, die aus einer Folge von zwei Fragezeichen und einem allgemein üblichen Sonderzeichen bestehen, z.B.: "??/" für "\".

Einige Hinweise auf Sprachelemente:

Namen von Funktionen, Variablen etc. können bei der CRAY X-MP aus bis zu 255 Zeichen bestehen, deren erstes ein Buchstabe sein muß (das "_" (underline) gilt als Buchstabe). Die folgenden Zeichen sind alphanumerisch. Andere Compiler beachten oft nur die ersten 8 Zeichen! Groß-/Kleinschreibung wird auch hier unterschieden.

Gültigkeit:

Die Definition von Variablen und Feldern gilt im allgemeinen nur innerhalb der jeweiligen Programmeinheit (Funktion), also lokal. Sollen Variable global verfügbar sein, müssen sie außerhalb von Funktionen definiert werden. Innerhalb einer Funktion verfügbar werden sie dann durch die Deklaration *extern*.

Speicherklassen:

Sie dienen dazu, festzulegen, was mit dem Speicherplatz von Variablen beim Übergang von einer Funktion zur anderen geschieht.

Klasse	Initialisierung*)	Erklärung
auto	u	lokale Gültigkeit innerhalb einer Funktion; der Wert wird beim Verlassen gelöscht.
register	u	wie *auto*; für häufigen Zugriff in schnellem Register abgelegt (die Anzahl solcher Variablen ist begrenzt).
static	0	lokale Gültigkeit; durch festen Speicherplatz bleibt der Wert erhalten.
extern	0	jede Funktion kann zugreifen, der Wert bleibt während des ganzen Programmlaufes erhalten.

*) Wert beim ersten Zugriff (automatische Initialisierung), u: undefiniert, 0: dezimal null.

Der Defaultwert der Klasse hängt von der Deklarationsart ab: für lokal deklarierte Variablen wird *auto* angenommen und für global deklarierte *extern*.

Datentypen:

Zur Ablage von Konstanten und Variablen unterschiedlicher Typen sind verschiedene Speicherlängen vorgesehen. Die Länge, d.h. die Anzahl der zur Darstellung verwendeten Bits, ist maschinenabhängig:

Typ	Erklärung	bit - Anzahl gemäß C (allgemein),	CRAY XMP	(Genauigk.)
int	einfach ganzzahlig	32	64	(46/64)
long	lang ganzzahlig	64	64	(64)
short	kurz ganzzahlig	16	64	(24)
unsigned	vorzeichenlos ganzzahlig	16	64	(24...64)
float	einfach Gleitkomma	32	64	(64)
double	doppelt genau Gleitkomma	64	64	(64)
[long double]	doppelte Länge von *double*	--	128	(128)
void	ohne Zahlenwert (z.B. Funktionsrückkehr)			
character	Zeichenvariable	8	8	(8)

Parameter-Übergabe (Datentransfer zwischen Funktionen):

Für lokal vereinbarte Variablen erfolgt eine Wertübergabe (keine Adreßübergabe), dadurch wird ihr Wert in der rufenden Funktion nicht verändert.

Bei global deklarierten Variablen wird ein Zeiger auf die Adresse der Variablen übergeben, und ihr Wert kann global verändert werden. Dieser Mechanismus wird auch für die Übergabe von Feldern angewandt; der Zeiger weist dann auf das erste Feldelement.

Rückgabe des Funktionswertes: Nach Abarbeitung der Funktion wird das Programm an der Stelle des Aufrufes fortgesetzt. Wird dort ein Wert (als Ergebnis der Funktion) erwartet, so wird der Wert der Variablen eingesetzt, die in der *return*-Anweisung angegeben wurde.

Beispiel:

`f=kreis(d)`	Aufruf der Funktion *kreis*, Parameter ist die Variable *d*
`kreis(durch)`	Anfang der Funktion, Name, Parameter
`float durch`	Definition und Typzuweisung der Parameter
`{`	
`float fl`	Def. und Typzuweisung lokaler Variablen
`fl=3.14/4*durch*durch`	
`return(fl)`	Rückkehr mit Übergabe des Wertes
`}`	von *fl* als Wert der Funktion

Im rufenden Programm steht jetzt auf der rechten Seite von *f=...* der Wert der Funktion *kreis*; dies ist per Anweisung der Wert der Variablen *fl*.

Für die Beschreibung weiterer Eigenschaften von C sei auf die Literatur verwiesen.

Die Anwendung der Möglichkeiten zur Interaktion mit FORTRAN und die Verwendung des Kommandos *cl* (Listing und Syntaxprüfung der C-Quelle) können in PCC z.Zt. nicht empfohlen werden.

Für SCC stehen etliche Makros und Utilities zur Verfügung, die helfen sollen, die vielfältigen Inkompatibilitäten zwischen C und FORTRAN sowie CAL (Assembler) zu überbrücken. Eine Beschreibung der Anwendungsmöglichkeiten soll eine in Vorbereitung befindliche Publikation enthalten (SN-3009).

Listen der Fehlermeldungen finden sich in den C-Reference-Manuals der jeweiligen Compiler. Die Meldungen von PCC beginnen mit CP oder CC, diejenigen von SCC mit SC, jeweils gefolgt von einer drei- oder vierstelligen Fehlernummer.

11. Fehlersuche in Programmen

11.1 Angaben zur Fehlersuche auf Shell-Ebene

Es gibt kein "Kochrezept" zur Fehlersuche. In diesem Kapitel werden nur verschiedene Hilfsmittel und Vorgehensweisen beschrieben, die, je nach Fall angewendet, das Einkreisen und Aufspüren eines Fehlers ermöglichen. Fehler können sich

- bereits während des Übersetzens
- erst bei der Programmausführung

bemerkbar machen. Wenn ein Kommando mit Fehler abbricht, wird der Shell-Variablen *exit status* in *$?* ein Wert ungleich Null zurückgegeben. Bricht ein Benutzerprogramm mit einem Fehler ab, wird ein Speicherabbild unter dem Namen *core* in den aktuellen Katalog abgelegt. Anschließend wird das nächste Kommando ausgeführt. Soll der Job beim Auftreten von gravierenden Fehlern abgebrochen werden, so muß am Jobanfang das Kommando

```
set -e
```

stehen. Bei Verwendung des ZIB-Standard-Prologs ist dies automatisch der Fall; ansonsten wird dringend empfohlen, dieses Kommando an den Anfang eines jeden Jobs zu stellen. Zusätzlich gibt es mehrere Möglichkeiten der gezielten Fehlerbehandlung:

Das "||"-Zeichen

Ein hinter diesem Zeichen stehendes Kommando wird nur ausgeführt, wenn das davorstehende mit Fehler abbricht.

Beispiel:

```
.
.
.
cft77 -esz myprog.f
segldr myprog.o
a.out || { cat myprog.l; debug; exit 1; }
.
.
.
```

In den ersten Zeilen wird das Programm *myprog.f* übersetzt und mit den Standardbibliotheken gebunden. Die Option *-esz* bedeutet, daß das Programm in die Übersetzerliste *myprog.l* geschrieben wird und Symboltabellen angelegt werden. So kann später den Variablen wieder ihr Name zugeordnet werden. Das fertige, ausführbare Programm bekommt den Namen *a.out*. In der letzten Zeile wird *a.out* ausgeführt. Läuft das Programm fehlerfrei, wird mit der nächsten Zeile fortgefahren. Andernfalls werden die Kommandos nach dem "||" ausgeführt:

- Die Übersetzerliste wird aufgelistet
- Das Hilfsmittel *debug* bereitet aus dem *core*-File und den Symboltabellen in *a.out* verständliche Informationen auf (siehe Kapitel 11.5.1).
- Mit *exit* wird der Job abgebrochen.

Nur mit *set -e* und ohne *exit* würde der Job nicht abgebrochen, da die Zeile 3 im obigen Beispiel als Einheit gesehen wird und insgesamt erfolgreich abgearbeitet wurde.

Die geschweiften Klammern dienen nur zum Zusammenfassen mehrerer Kommandos zu einer Einheit.

Das "&&" Zeichen

Ein hinter diesem Zeichen stehendes Kommando wird nur ausgeführt, wenn das davorstehende erfolgreich abgeschlossen wurde, d.h. der *exit status* Null ist ($? = 0).

Beispiel:

```
cft77 -es myprog.f
segldr myprog.o && a.out
```

a.out wird nur ausgeführt, wenn Übersetzen und Laden erfolgreich waren.

Flußkontrollbefehle auf Shell-Ebene

Damit ist eine sehr detaillierte Fehlerbehandlung möglich.

Beispiel (zur if-Anweisung vergl. 7.5.4):

```
if cft77 -esz myprog.f
then
    if segldr -o myprog myprog.o
    then
      if myprog
      then
        echo "well done"
      else
        mv core mycore
        debug -s myprog mycore
      fi
    fi
fi
```

In der ersten Zeile wird das Programm *myprog.f* übersetzt. Mittels *-e z* wird eine Symboltabelle angelegt. Durch die *if*-Anweisung wird das Folgende nur ausgeführt, wenn *$?* Null ist, die Übersetzung also erfolgreich war. Der Segmentlader bindet *myprog.o* mit den Standardbibliotheken zum ausführbaren Programm *myprog*. Der Parameter *-o* übergibt den Namen des gebundenen Programms an SEGLDR. Läuft *myprog* erfolgreich, wird in *stdout* "well done" ausgegeben. Im Fehlerfalle wird der *else*-Zweig durchlaufen. Zuerst wird das Speicherabbild von *core* in *mycore* umbenannt, um zu verhindern, daß es durch Absturz eines anderen Programms überschrieben wird. Anschließend werden mit dem Hilfsmittel *debug* die vorgefundenen Reste verständlich aufbereitet. Die Datei, die die Symboltabellen enthält, also das ausführbare Programm selbst, wird mittels der Option *-s* angegeben.

Die nächsten Beispiele zeigen andere Versionen des obigen Falls, hier mit C-Programmen:

```
cc -cg myprog.c
if [ $? -eq 0 ]
    then
    segldr -g -o myprog myprog.o
    if [ $? -eq 0 ]
      then
        myprog
      else
        debug -s myprog
    fi
fi
```

```
set -e              # exit if error
cc -g myprog.c      # compile and load C-program
if a.out            # run program ...
   then             # check
     true           # fine
   else             # error in program
     debug          # print erraneous stuff
fi                  # endif
```

Auch hier wird von *set -e* nur der gesamte *if - fi*-Block gesehen. Ein Fehler in *a.out* alleine beendet den Job also nicht.

Signals

Bei der Ausführung eines Prozesses senden verschiedene Ereignisse ein Signal *signal* an den Vaterprozess, meist also die Shell, was im Falle von Fehlern zum Abbruch des Prozesses führt. Signale haben eine Nummer (signal number); an die Shell gerichtete Signale können mit dem *trap*-Kommando abgefangen werden:

```
trap command-list signals
```

Falls eines der Signale von der Shell empfangen wird, wird erst *command list* und anschließend das nächste Kommando im Script ausgeführt. Einige Bibliotheksroutinen fangen die Signale selbst ab und führen eine Fehlerbehandlung durch; der *return code* ist dann ungleich Null.

Weitere Fehlermeldungen erscheinen mit einer Fehlernummer (keinem Signal), deren Bedeutung unter INTRO(2) im UNICOS SYSTEM CALLS REFERENCE MANUAL SR-2012 (siehe Anhang B.1) beschrieben ist. Wichtige Signalnummern daraus (die mit * bezeichneten erzeugen einen *core dump*):

0	Logout	nach *exit*-Kommando
1	SIGHUP	*Terminal Hangup*
2	SIGINT	*Terminal Interrupt* (*del* oder *break*)
3*	SIGQUIT	Abbruch eines Programms
4*	SIGILL	illegale Instruktion
5*	SIGTRAP	*Trace Trap*
6*	SIGABRT	*Abort*
7*	SIGERR	Fehlerausgang
8*	SIGFPE	*Floating Point Exception*
9	SIGKILL	Prozeßabbruch, nicht abfangbar
10*	SIGPRE	Programmbereichsüberschreitung
11*	SIGORE	Benutzerfeldüberschreitung
12*	SIGSYS	fehlerhaftes Argument zu einem *System Call*
13	SIGPIPE	schreiben auf eine *Pipe*, die nicht gelesen wird

14	SIGALRM	Alarm
15	SIGTERM	Prozeßabbruch mit Aufräummöglichkeit
16	SIGUSR1	Benutzer-definiertes Signal 1
17	SIGUSR2	Benutzer-definiertes Signal 2
18	SIGCLD	Ende eines *Child*-Prozesses
19	SIGPWR	Stromausfall
20	SIGMT	*Multitasking wake-up*
21	SIGMTKILL	*Multitasking* Programmabbruch
22	SIGBUFIO	FORTRAN asynchrones I/O-Ende
23	SIGRECOVERY	Prozeß-Recovery
24*	SIGUME	nicht korrigierbarer Speicherfehler
25*	SIGDLK	echter *Deadlock* (Multitasking)
26	SIGCPULIM	CPU-Zeitüberschreitung
27	SIGSHUTDN	System-Shutdown beginnt

11.2 Compiler-Optionen für CFT77 und CFT

Zur Erleichterung der Fehlersuche können Compiler mit bestimmten Optionen versehen werden. Bei FORTRAN-Compilern sind dies die Option *-m* zur Steuerung des Niveaus der ausgegebenen Meldungen und *-e* (enable) mit den Schlüsseln

a	*abort*
o	*array bounds checking*
q	*quit*
z	*debug*
D	*debug*

Einzelheiten dazu in den Kapiteln 8.1.3 (CFT77) und 8.3.3 (CFT).

11.3 Lader-Optionen

Es wird empfohlen, zur Fehlersuche folgende Optionen beim SEGLDR-Aufruf zu verwenden:

-f indef	belegt alle nicht initialisierten Teile eines Programms mit einem Wert, der zu einem Gleitkommafehler führt. Integer Variable erhalten dabei den Wert 7.009.149.132.560.400.384, ASCII-Variable den Wert "aE" (im ZIB voreingestellt).
-f ones	wirkt ähnlich, ein Wort wird mit binären Einsen gefüllt. Das ergibt keinen sinnvollen Gleitkommawert, für Integer-Variable ergibt sich -1, für ASCII-Variable kein druckbares Zeichen.

Wird der SEGLDR nur mittelbar, etwa durch *ld*, *cf*, *cf77* oder *cc* aufgerufen, kann diese Initialisierung mittels Durchreichen der Direktive

```
PRESET=INDEF      bzw.   PRESET=ONES
```

an den SEGLDR vorgenommen werden. Das Durchreichen geschieht beim *ld* mit der Option *-D dirstring*, bei den Übersetzern *cf*, *cf77* und *cc* mit der Option *-d dirstring*.

11.4 Hilfsprogramme zur Fehlersuche

11.4.1 Erzeugen eines statischen Aufrufbaumes, Semantikanalyse

Oftmals ist es nützlich, zu Beginn der Fehlersuche sich mittels eines statischen Aufrufbaumes eine Übersicht über die Struktur des jeweiligen Programms zu machen. Dazu dienen die UNICOS-Kommandos

```
ftref     (für Fortran-Programme)
cflow     (für C-Programme).
```

ftref erhält als Eingabe ein von einem FORTRAN-Compiler erzeugtes Listingfile; *cflow* die C- und, falls vorhanden, auch die YACC- und LEX-Dateien; beide Programme schreiben nach *stdout*.

Beispiel: Erzeugen eines Aufrufbaumes in

FORTRAN:

```
cft77 -exs prog.f
ftref -t full prog.l
```

C:

```
cflow -ix file
```

Das wertvollste Programm zur Entdeckung von Fehlern in C-Programmen ist das Standard-UNIX-Programm *lint*. Es nimmt eine Semantikanalyse vor und weist auf möglicherweise fehlerhafte, nicht portable oder "gefährliche" Konstruktionen hin.

Eine vollständige Beschreibung der Kommandos erhält man mit *man ftref*, *man cflow* bzw. *man lint*.

Im Unterschied zu diesen Kommandos arbeiten die nachfolgend beschriebenen Programme zur Laufzeit (SYMDUMP) bzw. nach fehlerhaftem Ende des Benutzerprogramms (debug).

11.4.2 debug

debug formatiert das Speicherabbild (coredump) zum Zeitpunkt des Programmfehlers. Dazu werden Symboltabellen benötigt, die bereits vom Compiler angelegt sein müssen. *debug* kann dann Speicheradressen den Namen aus dem Programm zuweisen. Benötigt werden das Speicherabbild, das im Fehlerfall automatisch mit dem Namen *core* angelegt wird, und die im ausführbaren Programm enthaltenen Symboltabellen.

Aufruf:

```
debug options corefile
```

Wichtige Optionen:

-s symfile	Datei mit der Debug Symbol Tabelle (Default: *a.out*).
-y s1,s2, ...	Liste der Symbole, die Debug nicht ausgeben soll (Default: alle Symbole).
-Y	listet keine Symbole
-b bl1,bl2, ...	Liste der COMMON-Blöcke, die Debug ausgeben soll (default: keine).
-B	listet alle COMMON-Blöcke
-d d1,d2, ...	Maximale Anzahl der Feldelemente in der jeweiligen Dimension (default: 20,5,2,1, ...).

Beispiel:

```
set -x
cft77 -eDm $HOME/prg.f
segldr prg.o
a.out || debug -B
```

debug sucht die Files *a.out* und *core*. Wurden diese umbenannt, müssen die Namen angegeben werden.

Beispiel:

```
cft77 -esz -b mybin myprog.f
segldr -o myexe mybin
myexe

.
.
.
mv core mycore        # Das File core wird umbenannt, um zu
                      # verhindern, daß es bei Absturz eines
                      # der nachfolgend aufgerufenen Pro-
                      # gramme überschrieben wird.
.
.
.
debug -s myexe mycore
```

11.4.3 SYMDUMP

SYMDUMP läßt sich aus einem laufenden Programm aufrufen und verhält sich wie *debug*. Voraussetzung für die Verwendung von SYMDUMP ist die Angabe der Option *-l db* im *segldr*-Kommando. *segldr* ergänzt diese Angabe zum vollen Pfadnamen */lib/libdb.a*; die Form *-l libdb.a* führt daher zu einem falschen Pfadnamen.

Beispiel:

```
INTEGER A, B
.
.
.
CALL SYMDUMP()
.
.
.
STOP
END
```

Eine genaue Beschreibung von *debug* und *SYMDUMP* findet sich im CRAY-Handbuch UNICOS Symbolic Debugging Package Reference Manual SR-0112 C (siehe Anhang B.1).

12. Einführung in die Optimierung der Rechenzeit

In der angewandten Mathematik werden Objekte durch arithmetische und logische Operationen verknüpft und in neue Objekte umgewandelt. Diese Objekte sind hauptsächlich Skalare (Zahlen), Vektoren und Matrizen.

Vektoroperationen lassen sich auf skalare Operationen mit den einzelnen Komponenten der Vektoroperanden zurückführen. Ebenso können Matrizenoperationen als Vektoroperationen mit den einzelnen Zeilen und Spalten der Matrizen beziehungsweise als skalare Operationen mit den einzelnen Elementen der Matrizen aufgefaßt werden.

Herkömmliche Rechner und die dafür entwickelten Programmiersprachen wie FORTRAN 77 bieten Darstellungen für Skalare (Variable), Vektoren und Matrizen (Felder oder Arrays), aber nur Operationen mit Skalaren. Vektor- und Matrizenoperationen müssen als skalare Operationen programmiert werden, z.B. in FORTRAN 77 durch DO-Schleifen (FORTRAN 8X wird die Formulierung von Operationen mit Feldern erlauben).

Im Abschnitt 12.1 wird vorgestellt, wie die CRAY-Architektur auf die effiziente Bearbeitung von Vektoroperationen ausgerichtet ist. Abschnitt 12.2 erläutert, wie Vektoroperationen auf der CRAY ablaufen und wie das Prinzip der Verkettung den gesamten Rechenvorgang beschleunigt.

In den folgenden Abschnitten werden dann die wichtigsten Programmiertechniken für die vektorielle Verarbeitung mit FORTRAN-Programmen zusammengestellt. Abschnitt 12.4 zeigt, wie die Autovektorisierer von CFT77 bzw. CFT in sequentiellen FORTRAN-Programmen vektorielle Operationen erkennen und wie sie vom Programmierer unterstützt werden können. Weitere Optimierungsmöglichkeiten werden in Abschnitt 12.5 vorgestellt. Abschnitt 12.6 weist auf die Bibliothek von optimierten Unterprogrammen hin.

Die Begriffe *multitasking*, *macrotasking*, *microtasking* und *autotasking* haben nichts mit der Optimierung der Rechenzeit zu tun. Diese Techniken dienen vielmehr dazu, die geforderte Leistung auf zwei oder mehr Teilaufträge (tasks) zu verteilen. Diese können dann mehrere vorhandene Prozessoren unabhängig voneinander ausnutzen.

12.1 Systemarchitektur der CRAY X-MP

Zum möglichst effizienten Einsatz der Rechenleistung sind grundlegende Kenntnisse über Hardware-Eigenschaften der CRAY X-MP des ZIB sehr nützlich.

Zwei identische Rechenwerke teilen sich den gemeinsamen Hauptspeicher und das Ein-/Ausgabesystem. Sie sind in Adreßteil, Skalarteil und Vektorteil mit jeweils eigenen Registern und Funktionseinheiten gegliedert.

Die wichtigsten Bestandteile und Arbeitsweisen der Rechenwerke werden in den folgenden Abschnitten erläutert. Weitergehende Fragen zur Hardware der CRAY X-MP können einem Band der Reihe *Lecture Notes in Computer Science* des Springer-Verlages entnommen werden: *K. A. Robbins, S. Robbins - The CRAY X-MP/Model 24 (1989)* (siehe Kapitel B.2).

12.1.1 Vektorteil des Rechenwerks

Zur Bearbeitung von Vektoren gibt es

8 V-Register mit je 64 Worten (je 64 Bit)
1 Vektorlängenregister (7 Bit)
1 Vektormaskenregister (64 Bit)

Die Vektorregister können blockweise vom Speicher oder elementweise aus S-Registern geladen werden.

Vektorregister, die als Operanden dienen, sind für die Dauer der Operation belegt. Vektorregister, die das Ergebnis einer Operation aufnehmen, können gleichzeitig als Operanden für die nächste Operation dienen. Die größte Wirkung dieser Verkettung (*chaining*) ergibt sich, wenn die zweite Operation beginnt, sobald das erste Ergebniselement der ersten Operation vorliegt.

Das Vektorlängenregister ermöglicht Operationen mit weniger als 64 Elementen. Vektoroperationen mit mehr als 64 Elementen werden in mehrere Operationen mit max. 64 Elementen zerlegt.

Jedes Bit des Vektormaskenregisters entspricht dem Element eines Vektorregisters. Es steuert die Auswahl der Elemente bei der vektoriellen Mischoperation (*vector merge*, siehe Kapitel 12.4.4: IF-Anweisungen).

Für verschiedene Operationen gibt es voneinander unabhängige Funktionseinheiten, die alle gleichzeitig arbeiten können. Sie erhalten ihre Operanden von den Registern und liefern ihr Ergebnis in einem anderen Register ab.

Alle Funktionseinheiten sind segmentiert, d.h. in Verarbeitungsstufen geteilt, die jeweils einen Maschinentakt für ihren Arbeitsanteil brauchen. Damit kann jede Funktionseinheit in jedem Takt das nächste Element des oder der Operanden entgegennehmen und nach einer für jede Operation festen Zahl von Takten in jedem Takt ein Ergebnis abliefern (pipelining).

Ist n die Zahl der Verarbeitungsstufen einer Funktionseinheit und l die Länge der zu bearbeitenden Vektoren, so vergehen vom Eintritt der ersten Vektorkomponente in die Pipeline bis zum Erscheinen des letzten Resultats $n+(l\text{-}1)$ Maschinentakte. Hinzu kommen jeweils drei Takte zur Vor- und Nachbereitung, so daß die Durchführung einer Vektoroperation insgesamt $n+l+5$ Takte erfordert.

Für die Vektorverarbeitung gibt es sieben Funktionseinheiten:

	Verarbeitungsstufen n (64-Bit-Operanden)
logische Verknüpfung	2
Verschiebeoperation	3 (für 128 Bit: 3 bis 4)
Zählen der gesetzten Bits	5
ganzzahlige Addition/Subtraktion	3
Gleitkommaaddition/Subtraktion	6
Gleitkommamultiplikation	7
reziproke Approximation	14

Die drei letztgenannten Funktionseinheiten für Gleitkommaoperationen werden auch für skalare Operationen benutzt.

Einfach genaue Gleitkommazahlen bestehen aus einem Vorzeichenbit, 15 Bit Exponent zur Basis 2 und 48 Bit Mantisse (siehe Anhang C.2). Die Rechenergebnisse sind immer normalisiert. Für doppelt genaue Berechnungen (95 Bit Genauigkeit) ist keine spezielle Hardware vorgesehen. Sie werden softwaremäßig durchgeführt und sind deutlich zeitaufwendiger (siehe 12.5.3).

Die ganzzahlige Multiplikation wird unter Verwendung von Gleitkommamultiplikationen, Verschiebeoperationen sowie (in Abhängigkeit von der Größe der Operanden) eventuell auch von einer Additionsoperation durchgeführt.

Zur Berechnung eines Kehrwertes werden von der Funktionseinheit für reziproke Approximation drei Iterationen eines Newton-Verfahrens durchgeführt, ausgehend von einem approximierten Tabellenwert. Das auf 30 Bit genaue Ergebnis wird durch eine vierte Newton-Iteration, die in der Multiplikationseinheit durchgeführt wird, auf volle Genauigkeit gebracht.

Gleitpunktdivision geschieht durch Multiplikation mit dem Kehrwert (skalar: 29 Takte; vektorisiert: alle drei Takte ein Resultat nach einer Startup-Zeit von 38 Takten).

12.1.2 Skalarteil des Rechenwerks

Der Skalarteil zum Rechnen mit Zahlen besteht aus

8 S-Registern (je 64 Bit) zum Rechnen
64 T-Registern (je 64 Bit) als Zwischenspeicher

und vier Funktionseinheiten

Addition/Subtraktion (3 Takte)
Verschiebeoperation (2 Takte)
logische Verknüpfung (1 Takt)
Zählen von gesetzten bzw. nicht gesetzten Bits (4 bzw. 3 Takte)

Die Additionseinheit verarbeitet ganzzahlige 64-Bit-Werte ohne Überlauf. Eine Multiplikationseinheit für ganzzahlige 64-Bit-Werte gibt es nicht.

Für Gleitkommaoperationen mit Skalaren werden die Funktionseinheiten des Vektorteils mitbenutzt.

12.1.3 Adreßteil des Rechenwerks

Für Adreß- und Indexrechnungen sowie Schleifenkontrollen stehen zwei Sätze von Registern

8 A-Register (je 24 Bit) zum Rechnen
64 B-Register (je 24 Bit) als Zwischenspeicher

und zwei Funktionseinheiten

Addition/Subtraktion (2 Takte)
Multiplikation (4 Takte) zur Verfügung.

Die beiden Funktionseinheiten bieten ganzzahlige 24-Bit-Arithmetik auf den A-Registern ohne Überlauferkennung.

12.1.4 Kommunikation der Rechenwerke untereinander

Die Prozessoren der CRAY X-MP können über gemeinsame Register oder über den gemeinsamen Hauptspeicher kommunizieren. Allen Rechenwerken stehen drei identische Sätze gemeinsamer Register zur Verfügung. Jeder Satz besteht aus

8 gemeinsamen Adreßregistern (je 24 Bit)
8 gemeinsamen Skalarregistern (je 64 bit)
32 gemeinsamen *Semaphore*-Registern (je 1 Bit)

Diese Register werden unter anderem auch beim *microtasking* und *autotasking* verwendet. Weiterhin haben die Prozessoren Zugriff auf ein gemeinsames Register, das die Maschinentakte zählt und somit genaueste Laufzeitmessungen ermöglicht.

12.1.5 Befehlspuffer und Befehlsabarbeitung

Jedes Rechenwerk hat vier Befehlspuffer für je 128 aufeinanderfolgende Befehlspakete zu je 16 Bit (=32 Worte). Unter optimalen Bedingungen kann mit jedem Maschinentakt ein Befehl an eine der drei Recheneinheiten (Skalar-, Vektor- oder Adreßteil) abgesetzt werden. Wenn der nächste auszuführende Befehl in einem anderen Puffer als der vorhergehende liegt, tritt eine Verzögerung von zwei Takten ein. Liegt der nächste Befehl nicht im Puffer, wird ein Puffer ausgewählt und vom Speicher aus neu geladen. Bis der Befehl dann vorliegt, vergehen 16 bis 19 Takte. Sprünge vorwärts oder rückwärts innerhalb eines Puffers bedeuten keine Verzögerung. Befehle werden erst dann abgesetzt, wenn die zugehörigen Operanden in Registern vorliegen, da die Funktionseinheiten nur auf Registern operieren.

12.1.6 Hauptspeicher

Der Hauptspeicher von vier Millionen Worten zu je 72 Bit (64 Datenbits und 8 Prüfbits SECDED, *single error correction and double error detection*) hat eine Zykluszeit von vier Maschinentakten (=38 ns). So lange ist eine Speichereinheit belegt für den Transport eines Wortes.

Zugriffszeiten für Transporte vom Speicher in die Register:

14 Takte	für Adressen und Skalare
17 + Vektorlänge	für Vektoren
16 + Blocklänge	für Blockübertragung in B- und T-Register

Maximale Übertragungsraten:

B-, T- und V-Reg.	1 Takt für 3 Worte
A- und S-Register	2 Takte für 1 Wort
Befehlspuffer	4 Takte für 128 Pakete (=32 Worte)
Ein-/Ausgabe	1 Takt für 2 Worte

Organisation des Hauptspeichers

Der Hauptspeicher ist in vier Abschnitte zu je acht Bänken geteilt. Jedes der beiden Rechenwerke hat je eine Verbindung zu jedem der vier Abschnitte:

Im Abschnitt 0 liegen die Bänke 0,4,8,...,28; Im Abschnitt 1 liegen die Bänke 1,5,9,... usw.

Jedes Rechenwerk hat vier Zugriffspfade zum Speicher, nämlich drei für die Registerübertragung:

A	Vektor- und B-Register laden
B	Vektor- und T-Register laden
C	Skalare laden und alle Speicheroperationen

und einen davon unabhängigen Pfad für Ein- und Ausgabe.

Gleichzeitiges Laden und Speichern von Vektoren oder Blöcken (bidirectional memory mode) kann zu unerwünschten Überlappungen und Fehlern führen, die (beim Programmieren im Assembler CAL) durch die Programmierung vermieden werden müssen. Normalerweise verhindern die FORTRAN-Übersetzer diese Überlappungen (siehe Kapitel 12.4.4).

Für skalare Übertragungen müssen die drei Pfade A, B und C frei sein, um die korrekte Reihenfolge von Blockübertragungen und Skalarzugriffen zu garantieren..

Wenn Befehle zu laden sind, werden alle acht Verbindungen zum Hauptspeicher gesperrt (also für beide Rechenwerke): nach spätestens drei Takten sind die laufenden Vorgänge abgeschlossen und in den nächsten vier Takten werden dann über die acht Verbindungen 32 Worte von allen 32 Bänken in einen Befehlspuffer geladen.

Speicherzugriffskonflikte

In der CRAY X-MP sind drei Arten von Speicherzugriffskonflikten möglich:

Bank belegt: Eine Bank ist bei einem Zugriff 4 Takte belegt. Wenn ein Zugriffspfad eine noch aktive Bank ansprechen will, werden alle Pfade dieses Rechenwerks für ein, zwei oder drei Takte angehalten, bis die Bank wieder frei ist.

Gleiche Bank: Wenn zwei Pfade aus verschiedenen Rechenwerken dieselbe Bank ansprechen, muß ein Rechenwerk einen Takt warten. Anschließend liegt ein Bankbelegtkonflikt vor.

Gleicher Abschnitt: Wenn zwei Pfade aus einer CPU Bänke in demselben Abschnitt ansprechen, muß ein Pfad einen Takt warten.

Welche Pfade und Rechenwerke jeweils zu warten haben, wird durch geeignete Prioritäten geregelt.

12.2 Prinzip der Vektorverarbeitung

Ein *Vektor* im Sinne der CRAY ist eine Ansammlung von Worten, die mit konstantem Abstand im Hauptspeicher abgelegt sind. Es ist dabei gleichgültig, ob es sich um Vektoren im mathematischen Sinne handelt. Beispiele für Vektoren sind eindimensionale Felder oder (bei konventioneller Abspeicherung) Spalten, Zeilen und Diagonalen einer Matrix.

Beispiel (in FORTRAN):

```
      REAL X(0:500), A(17,0:300), B(0:100)
      DO 10 I = 0,100,2
10    X(5*I) = A(J,3*I) * B(I)
```

Die Elemente B(0), B(2), ... B(100) wie auch X(0), X(5), ... X(500) und A(J,0), A(J,6) ... A(J,300) bilden jeweils einen *Vektor* der Länge 51. Die *Vektorlänge* ist im allgemeinen nicht gleich der Dimension der Felder, sondern gleich der Anzahl der Schleifendurchläufe. Die *konstanten Abstände* sind 2 bei B, 2*5 = 10 bei X und 2*3*17 bei A (Faktor 17 aufgrund der FORTRAN-Konvention, Elemente mehrdimensionaler Felder so im Hauptspeicher abzulegen, daß sich benachbarte Elemente im linken Index um eins unterscheiden, d.h. linksstehende Indices zuerst hochgezählt werden). Der konstante Abstand, mit dem Vektorelemente im Speicher abgelegt sind, ergibt sich somit aus der Schrittweite der DO-Schleife, den Indexausdrücken, die die Laufvariable enthalten, und den Felddimensionen.

Auf solchen Vektoren können im Vektorrechner *Vektoroperationen* ausgeführt werden.

Beispiel:

Es sollen zwei Vektoren A und B der Länge N addiert werden und das Ergebnis dem Vektor X zugewiesen werden:

```
      DO 10 I = 1,N
10    X(I) = A(I) + B(I)
```

Ausführung auf herkömmlichen Rechnern:

> Führe folgende Befehle N mal aus:
> Lade je eine Komponente von A und B in (skalare) Register;
> addiere im Rechenwerk die Inhalte der beiden Register;
> speichere den Inhalt des Ergebnisregisters in der zugehörigen Komponente von X.

Ausführung auf Vektorrechnern ohne Vektorregister:

> Addiere die Komponenten der Vektoren in einer Pipeline.

Ausführung auf Vektorrechnern mit Vektorregister, z.B. CRAY:

> Lade die Vektoren A und B in Vektorregister;
> addiere die Komponenten der Vektoren in einer Pipeline und sammle die Resultate in einem dritten Vektorregister;
> speichere den Inhalt des Ergebnisregisters in Vektor X.
>
> Sind die Vektoren länger als die Vektorregister (Länge L), so werden die Vektoren in Abschnitten der Länge L bearbeitet, d.h. die drei Befehle entsprechend oft ausgeführt (s.u.).

Vorteile der Vektoroperationen:

Es sind im allgemeinen viel weniger Befehle aus dem Hauptspeicher zu holen, zu dekodieren und auszuführen. Die Verarbeitung der Daten geschieht in Pipelines, die nach einer gewissen Startup-Zeit mit jedem Maschinentakt ein Resultat liefern.

Effizienter Einsatz des Vektorrechners setzt voraus, daß ein signifikanter Anteil der Daten auf Vektoren und die zugehörigen Operationen auf Vektoroperationen abgebildet werden. Dies ist in erster Linie Aufgabe des Compilers, doch kann der Anwender durch geeignete Programmierung sehr viel dazu beitragen.

12.2.1 Vektoroperationen auf der CRAY

In CRAY-Rechnern werden alle Rechenoperationen auf Daten ausgeführt, die sich in Registern befinden. Dies hat den Vorteil, daß Funktionseinheiten und Hauptspeicher entkoppelt sind: die Funktionseinheiten können wesentlich schneller arbeiten, als Zugriffe auf Daten im Hauptspeicher möglich sind. Ziel vieler Optimierungen ist daher, das Volumen der zwischen Hauptspeicher und Vektorregistern transportierten Daten zu minimieren durch möglichst häufige Wiederverwendung der Daten in den Vektorregistern. Die Vektorregister der CRAY haben die Länge 64. Ist im letzten Beispiel N < = 64, so erzeugt der Compiler neben den Lade- und Speicherbefehlen nur einen Vektorbefehl (in FORTRAN 8X-Schreibweise):

```
      X(1:N) = A(1:N) + B(1:N)
```

Ist N > 64 oder ist der Wert von N zur Compilationszeit noch nicht bekannt, müssen die Vektoren in Abschnitten der maximalen Länge 64 bearbeitet werden (wieder in FORTRAN 8X-Schreibweise):

```
      I1= 1
      I2= MOD (N-1, 64) + 1
      DO 10 K = 1, (N + 63)/64
            X(I1:I2) = A(I1:I2) + B(I1:I2)
      I1= I2 + 1
      I2= I1 + 63
10    CONTINUE
```

Die abschnittweise Bearbeitung langer Vektoren bleibt dem Benutzer verborgen, sofern er in einer höheren Programmiersprache programmiert. Es genügt, in Vektorbefehlen der Art X(1:N) = A(1:N) + B(1:N) zu denken.

Entsprechend den Funktionseinheiten des Vektorteils sind folgende Operationen hardwaremäßig realisiert:

für beliebige Vektoren Bit-Operationen:
- Verschieben
- Zählen gesetzter Bits
- bitweise logische Verknüpfung

und für ganzzahlige Vektoren:
- Addition und Subtraktion

sowie für Vektoren mit Gleitkommazahlen:
- Addition und Subtraktion
- Multiplikation
- Approximation des Kehrwerts

Auf Grundlage dieser Hardwarefunktionen sind viele weitere Basisfunktionen softwaremäßig als Vektoroperationen realisiert, z.B. ganzzahlige Multiplikation, Division von ganzen Zahlen und Gleitkommazahlen, schließlich auch fast alle Intrinsic-Functions in FORTRAN. Für die Konvertier- und logischen Funktionen wird *inline*-Code erzeugt, für die mathematischen Funktionen (SQRT, EXP, SIN, COS, ...) dagegen Unterprogrammaufrufe. Die komplexeren der softwaremäßig realisierten Vektoroperationen benötigen mehrere Takte pro Resultat.

12.2.2 Wichtige Begriffe der Vektorverarbeitung

Pipeline

Die Komponenten eines Vektors werden in der CRAY nicht parallel im Sinne von gleichzeitig verarbeitet. Zentral für das Verständnis der Vektorverarbeitung ist der Begriff der Pipeline und damit die Segmentierung der Funktionseinheiten.

Beispiel:

Wir betrachten die Vektoraddition von Gleitkommazahlen. Die Addition einer Vektorkomponente sei der Einfachheit halber in vier Verarbeitungsstufen (Segmente) unterteilt - tatsächlich sind es sechs:

Vergleich der Exponenten,
Angleichung der Mantissen,
Addition der Mantissen und
Normalisierung.

Zunächst werden die Exponenten der jeweils ersten Komponenten verglichen. Im nächsten Takt werden die Mantissen entsprechend angepaßt und gleichzeitig die Exponenten der jeweils zweiten Komponenten verglichen.

Wenn im vierten Takt die ersten Komponenten normalisiert werden, können bereits die Exponenten der vierten Komponenten verglichen werden. Nach dieser Anlaufzeit liefert das Rechenwerk mit jedem Maschinentakt eine Ergebniskomponente.

Verkettung (chaining)

Die von einer Funktionseinheit im Vektorteil des Rechenwerks erzeugten Resultate können als Eingabedaten in einer anderen Funktionseinheit verwendet werden, bevor die erste Vektoroperation beendet ist (chaining). Auf diese Weise können die unabhängigen Funktionseinheiten der Vektoreinheit zu längeren Pipelines verkettet werden. Auch das Laden oder Speichern von Vektorregistern kann mit Vektoroperationen verkettet werden.

Beispiel: Die Vektoroperation

```
X(1:N) = ABS (A(I,1:N) * B(1:N) + C(1:N))
```

kann durch Verkettung von

Laden der Vektorregister
Multiplikation
Addition
Bildung des Absolutbetrages
Speichern

in einer langen Pipeline durchgeführt werden, die nach einer gewissen Anlaufzeit mit jedem Takt ein Resultat abliefert. Wenn die ersten Vektorkomponenten geladen werden, können sie gleichzeitig in die erste Funktionseinheit, hier die Multiplikation, eintreten. Sobald die Komponenten eine Funktionseinheit verlassen, können sie in einer anderen Funktionseinheit, hier der Addition, weiterverarbeitet oder abgespeichert werden, während immer noch die weiteren Komponenten der ersten Operanden geladen werden.

12.3 Optimierungsstrategien bei der CRAY

CRAY-Rechner sind aus Funktionsgruppen mit sehr unterschiedlichen Arbeitsweisen aufgebaut, die sich zum Teil um Größenordnungen in ihrer Arbeitsgeschwindigkeit unterscheiden. Ziel einer Optimierung ist, die schnellsten Teile des Rechners möglichst ausschließlich und effektiv zu nutzen. Hinsichtlich der Rechenoperationen bedeutet dies die Nutzung des Vektorteils im Rechenwerk, hinsichtlich der Datenspeicherung Nutzung der obersten Stufen der Speicherhierarchie (siehe Kapitel 1.1.2), d.h. der Register und des Hauptspeichers. Die effektive Nutzung dieser Funktionsteile wird in den nachfolgenden Abschnitten besprochen.

Die wichtigsten Vorbedingungen für einen effizienten Einsatz des Vektorrechners leiten sich aus diesen Überlegungen ab, nämlich -

* die Möglichkeit, signifikante Anteile der Daten auf (lange) Vektoren und die Rechenoperationen auf Vektoroperationen abzubilden;
* die Möglichkeit, alle Daten gleichzeitig im Hauptspeicher zu halten, oder alternativ umfangreiche und häufige I/O-Vorgänge mit Rechenoperationen zu überlappen.

Die Verantwortung für die Abbildung der Daten auf Vektoren liegt beim Anwender: durch die Wahl geeigneter Algorithmen (oftmals solcher, die auf skalaren Rechnern als nicht optimal gelten) und durch geschickte Programmierung kann der Anteil an Vektoren und Vektoroperationen in den meisten Fällen drastisch gesteigert werden. Die Erkennung von Vektoroperationen, also unabhängigen Operationen auf Vektorkomponenten, wie auch deren Übersetzung in Vektorinstruktionen leisten im wesentlichen die "autovektorisierenden" Compiler. Deren Fähigkeiten sind jedoch begrenzt. Der Anwender sollte daher die Compiler beim Vektorisieren unterstützen durch

* geeignete Programmierung, die dem Compiler die Erkennung vektorisierbarer Programmteile erleichtert;
* Verwendung von Compiler-Direktiven, insbesondere zur Kennzeichnung vektorisierbarer Schleifen, die der Compiler nicht als solche erkennt;

* und (derzeit nur bei FORTRAN-Programmen) Durchführung einer Abhängigkeitsanalyse mit dem Präprozessor *fpp*, der vektorisierbare Schleifen bzw. definitiv nicht vektorisierbare Schleifen durch entsprechende Compiler-Direktiven kennzeichnet.

Nun zum zweiten Kernpunkt, der Minimierung und Beschleunigung von Zugriffen auf den Hintergrundspeicher. Auf Grund der sehr begrenzten Hauptspeicherkapazität (32 MB = 4 MW) der CRAY X-MP des ZIB muß bei vielen Anwendungen auf vergleichsweise langsame Hintergrundspeicher (Erweiterungsspeicher SSD, Plattenspeicher) zurückgegriffen werden. Die Zahl solcher Zugriffe läßt sich minimieren durch

* Wahl geeigneter Algorithmen
* geeignete Programmierung (z.B. Einsparung von Hauptspeicherplatz durch Mehrfachverwendung von Feldern).

Die Beschleunigung unvermeidlicher Zugriffe auf Hintergrundspeicher bzw. (transparent für den Anwender) auf Zwischenpuffer läßt sich erreichen durch

* Benutzung des schnellsten Hintergrundspeichers (SDS bei Scratch-Dateien) und günstiger I/O-Methoden (unformatiert!);
* Kohärenz der Daten: die Daten aufeinanderfolgender Zugriffe sollten auf dem Hintergrundspeicher möglichst dicht beieinander liegen, so daß schnellere Zwischenpuffer zur Auswirkung kommen;

Ferner kann die Verweilzeit eines Prozesses/Programms verkürzt werden durch

* Verwendung asynchroner I/O-Methoden zur Überlappung von I/O- und Rechenoperationen.

Zur Beschleunigung von I/O-Operationen ist die CRAY mit einem Erweiterungsspeicher (SSD) ausgerüstet. Vom gesamten Erweiterungsspeicher (128 MW) stehen Anwendungsprogrammen zwei Bereiche zur Verfügung: der SDS (11MW) als Ergänzung des Hauptspeichers sowie der *ldcache* (105 MW) als schneller Pufferspeicher zwischen Hauptspeicher und Plattenspeicher. Letzterer wird automatisch bei allen I/O-Operationen im */tmp* genutzt. Die Verwendung des SDS muß vom Benutzer selbst initiiert werden.

Sind die Optimierungsmöglichkeiten hinsichtlich der Vektorisierung und der Benutzung von Hintergrundspeicher ausgeschöpft, lohnt es sich in vielen Fällen, weitere Optimierungen zu versuchen (fine tuning), nämlich die Minimierung und Beschleunigung der Hauptspeicherzugriffe. Verminderung der Hauptspeicherzugriffe ist möglich durch

* geeignete Programmierung (der Datenfluß zwischen Hauptspeicher und den Registern ist durch die vom Compiler vorgenommene Registerbelegung festgelegt; diese Belegung kann innerhalb gewisser Grenzen auch beim Programmieren in einer Hochsprache beeinflußt werden. Beispielsweise erleichtert die Verwendung weniger komplexer statt mehrerer einfacher arithmetischer Ausdrücke die Wiederverwendung von Registerinhalten).
* günstige Wahl der Befehlspuffergrenzen (mittels der Compiler-Direktive CDIR$ ALIGN beim CFT) z.B. so, daß sich alle Befehle einer Schleife in einem Befehlspuffer befinden.

Wegen der relativ langen Totzeiten von Speicherzellen nach einem Zugriff ist der Hauptspeicher in Bänke (32 auf der CRAY X-MP) aufgeteilt. Je nachdem, wie lange der letzte Zugriff auf eine Bank zurückliegt, kann ein Hauptspeicherzugriff unterschiedlich lange dauern. Beschleunigung von Hauptspeicherzugriffen wird erreicht durch

* Vermeidung von sogenannten Bankkonflikten durch geeignete Programmierung.

12.4 Vektorisierung von FORTRAN-Programmen

Hinweis: Die in diesem Abschnitt zusammengestellten Erläuterungen gelten im wesentlichen gleichermaßen für CFT77 3.1 und CFT 1.16. Generell empfiehlt es sich, den CFT77 zu verwenden. Er erzeugt in den meisten Fällen schnelleren Code. Hinzu kommt, daß der CFT nicht mehr weiterentwickelt wird; die Version 1.16 ist definitiv die letzte Version. Ab 1. Januar 1991 wird die Unterstützung seitens CRAY für diese Compilerlinie ganz eingestellt. Aussagen, die nicht für beide Compiler gelten, sind jeweils besonders gekennzeichnet.

Standard-FORTRAN 77 bietet keine Sprachmittel für Vektoroperationen, für FORTRAN 8X sind solche vorgesehen. Im CRAY-FORTRAN-Compiler CFT77 wurde ein Teil dieser Spracherweiterungen schon eingeführt. Aus der Erkenntnis, daß Vektoroperationen in FORTRAN 77 komponentenweise - häufig in DO-Schleifen - programmiert sind, versucht der Autovektorisierer von CFT77 bzw. CFT, aus DO-Schleifen Vektoroperationen herauszulesen. Ist er erfolgreich, so sagt man, die Schleife *vektorisiert*. In einigen Zweifelsfällen, die erst zur Laufzeit entscheidbar sind, erzeugen die Übersetzer sowohl skalaren als auch vektoriellen Code mit einer Abfrage, die beim Ablauf die sequentielle oder die vektorielle Variante auswählt. Man spricht dann von *bedingter Vektorisierung*.

Vorteil dieses Vorgehens, also des Verzichts auf über den FORTRAN-Standard hinausgehende Sprachmittel für Vektoroperationen, ist die Portabilität, d.h. daß in Standard-FORTRAN 77 (sequentiell) geschriebene Programme die Vektoreigenschaften der CRAY nutzen können und weiterhin auf sequentiellen Rechnern ablauffähig sind. Nachteilig ist der Umstand, daß es schwierig bis unmöglich sein kann, in den sequentiellen Anweisungen automatisch Vektoroperationen zu erkennen und die volle Leistungsfähigkeit der CRAY zu nutzen. Zur Minderung dieses Nachteils bietet CRAY einerseits eine Reihe von optimierten Routinen (siehe Kapitel 12.6) und andererseits im CFT77 bzw. CFT Möglichkeiten, mit denen der Programmierer den Autovektorisierer unterstützen kann. Sie werden in den folgenden Abschnitten vorgestellt.

12.4.1 Vorgehensweise bei der Optimierung

Ziel bei der FORTRAN-Programmierung an der CRAY ist, einen hohen Anteil an vektorisierbarem Code mit möglichst langen Vektoren zu erreichen, um hohe Verarbeitungsgeschwindigkeiten zu erzielen. Folgende Vorgehensweise ist dabei sinnvoll:

a) Das Programm muß richtig laufen, d.h. die richtigen Ergebnisse liefern.

b) Lasse das FORTRAN-Programm von dem Präprozessor *fpp* analysieren und eventuell modifizieren (siehe Kap. 8.5).

c) Lasse den Autovektorisierer von CFT77 bzw. CFT innere DO-Schleifen vektorisieren. Er gibt Meldungen aus, wenn sie nicht vektorisierbar sind.

d) Stelle mit FLOWTRACE fest, in welchen Routinen die meiste Rechenzeit verbraucht wird (siehe Kapitel 8.1.3, Option -ef).

e) Analysiere nichtvektorisierte DO-Schleifen.

f) Unterstütze den Autovektorisierer durch Optionen und Direktiven.

g) Verwende geeignete optimierte Routinen aus SCILIB (siehe Kapitel 12.7).

h) Prüfe andere Optimierungsmöglichkeiten, z.B. Abrollen von Schleifen (nur CFT, siehe Kap. 8.3.2) und *inline expansion* (nur CFT77, siehe Kapitel 8.1.2 und Kapitel 12.5, sowie in Verbindung mit *fpp* in Kapitel 8.5 und 12.4.).

Wenn diese Maßnahmen zu keinem befriedigenden Ergebnis führen:

i) Überarbeite das Programm und verwende z.B. einen anderen Algorithmus oder eine Struktur, die sich besser zur vektoriellen Verarbeitung eignen.

Generell ist zu bedenken:

j) Nutze möglichst vorhandene, optimierte Software.

12.4.2 Was vektorisieren CFT77 bzw. CFT automatisch?

Welche Schleifen vektorisierbar sind, ergibt sich rein mathematisch aus einer Datenabhängigkeitsanalyse. Der FORTRAN-Präprozessor *fpp* führt solch eine Analyse durch (die Vektorisierungsinformation wird in Form von Compiler-Direktiven im Quellcode niedergelegt); die Übersetzer CFT77 und insbesondere CFT gehen weniger analytisch vor und arbeiten zum Teil mit heuristischen Prinzipien. Die Analysewerkzeuge in den Übersetzern werden ständig verbessert, so daß sich die Kriterien zur Entscheidung, ob ein Übersetzer Vektoroperationen erkennen kann, mit der Zeit ändern. Die Rangordnung der FORTRAN-Übersetzer hinsichtlich ihrer Fähigkeit, Vektoroperationen zu erkennen, ist derzeit

fpp,
CFT77,
CFT,

wobei *fpp* von FORTRAN nach FORTRAN übersetzt. Generell ist die Kombination *fpp* und *CFT77* zu empfehlen, zumal der *CFT* ab 1991 von CRAY nicht mehr unterstützt wird. Das Schwergewicht in den nachfolgenden Erläuterungen liegt daher bei *CFT77* und *fpp*.

Die folgende Aufstellung enthält Bedingungen, die für CFT77 3.1 bzw. CFT 1.16 erfüllt sein müssen, damit DO-Schleifen vektorisiert werden können (die auftretenden Begriffe werden nach dem Beispiel erläutert):

* Nur innere DO-Schleifen sind vektorisierbar.
* Mindestens eine Vektorreferenz muß in der Schleife links von einem Gleichheitszeichen stehen oder es muß eine Vektorreduktion vorliegen.
* Alle Variablen sind Vektorreferenzen, CIVs, Temporäre Vektoren, Invarianten oder Pseudovektoren.
* Die arithmetischen Ausdrücke verwenden REAL- (64 Bit), DOUBLE PRECISION (128 Bit), INTEGER- (46 Bit) und COMPLEX-Arithmetik (zweimal 64 Bit).
* Alle Aufrufe von FORTRAN 77 Intrinsic Funktionen mit Vektorversion (SIN, COS, ABS, MAX, SQRT usw.) haben vektorisierbare Ausdrücke als Argumente.

Einfache IF-Anweisungen wie bedingte Zuweisungen, Suchschleifen und bedingte Blöcke (IF-THEN-ELSE) verhindern die Vektorisierung im allgemeinen nicht (siehe Kapitel 12.4.4).

Anweisungsfunktionen werden durch die entsprechenden Ausdrücke mit den aktuellen Parametern ersetzt. Die Vektorisierbarkeit der resultierenden Anweisung wird mit obigen Regeln überprüft.

Beispiel für eine vektorisierbare Schleife:

```
      S = 0.
      K = 200
      DO 1 M = 100,200,3
            A(M) = 3.5                    ! Feldelement mit CIV M
            B(K) = A(M)+SIN(A(M))         ! SIN ist vektorisierbare Funktion
            T = C(K) ** 2 + 1.2           ! T ist Temporärer Vektor
            S = S + C(K)                  ! Vektorreduktion
            K = K - 1                     ! K ist CIV
            A(M-1) = 23.3+SQRT(T)         ! SQRT ist vektorisierbare Funktion
            E(M,K) = 0.0                  ! E ist Pseudovektor, da mit zwei CIVs
                                          ! indiziert
1     CONTINUE
```

Invariante

Invariant heißt eine Konstante oder eine Variable, die in einer Schleife benutzt aber nicht verändert wird.

Beispiel:

```
      DO 20 I = 1, 100
20          A(I) = B(I) + X*Z**K          ! X, Z und K sind Invariante
```

CIV (constant increment variable)

Die Elemente eines für Vektoroperationen geeigneten Feldes müssen im Speicher mit konstantem Abstand (increment) abgelegt sein. Zur Adressierung dieser Felder sind daher nur Variable geeignet, deren Wert bei jedem Durchlauf der Schleife um einen invarianten Ausdruck erhöht oder vermindert werden. Solche Variablen vom Typ INTEGER oder REAL werden als CIV bezeichnet. Beim CFT gibt es hinsichtlich der Definition von CIV's einige Einschränkungen (siehe Kapitel 10.1.2 im CFT Reference Manual).

Beispiele:

```
      DO 10 I = 10,30
            J = J - 8
            K = 4 + M * I
10    A(I) = B(J) * C(K)                  !I, J und K sind CIVs, M ist invariant
                                          ! und die Schleife vektorisiert.
                                          ! indiziert

      DO 20 I = 10,30
            J = J - I
            K = 4 + K * I
20    A(I) = B(J) * C(K)                  ! I ist CIV, aber J und K verändern
                                          ! sich mit wachsenden Abständen und
                                          ! verhindern die Vektorisierung.
```

Temporärer Vektor (scalar temporary)

Temporäre Vektoren sind Variable, die die folgenden Bedingungen erfüllen: sie werden innerhalb einer Schleife nur einmal durch einen vektorisierbaren Ausdruck definiert und führen, wenn sie an weiteren Stellen in der Schleife durch den definierenden Ausdruck ersetzt werden, auch dort zu vektorisierbaren Anweisungen.

Temporäre Vektoren werden in einem Vektorregister gehalten. Am Schluß der Schleife wird der Wert der letzten Komponente in der skalaren Variablen gespeichert.

Beispiel:

Die Schleife

```
      DO 70 I = 50, 10, -3
            S = C(K) + 4.
            T = A(I) + 1.35
            B(I) = X - T
            D(I) = SIN( S + T - B(I) )
70    CONTINUE
```

wirkt wie

```
      DO 70 I = 50, 10, -3
70    B(I) = X - A(I) - 1.35

      DO 71 I = 50, 10, -3
71    D(I) = SIN( C(K) + 5.35 + A(I) - B(I) )
```

Im obigen Beispiel sind T und S Temporäre Vektoren. Die Schleife wird vektorisiert.

Vektorreferenz (vector array reference)

Als Vektorreferenz wird beim CFT eine Feldreferenz bezeichnet, bei der ein Indexausdruck eine CIV enthält und alle weiteren Indexausdrücke invariant sind. Der Indexausdruck, der die CIV enthält, muß ein linearer Ausdruck sein, der in die Form

+- invarianter Ausdruck1 * CIV +- invarianter Ausdruck2

überführt werden kann, wobei *Ausdruck1* nur Multiplikationen enthalten darf. Es gibt eine Reihe weiterer Einschränkungen hinsichtlich der möglichen Form von *Ausdruck1* und *Ausdruck2* (siehe Kapitel 10.1.2 im CFT-Reference-Manual). Beim CFT77 werden alle Feldreferenzen mit veränderlichem Indexausdruck als Vektorreferenz bezeichnet. Doch auch hier gilt, daß in der CIV lineare Indexausdrücke für die Vektorisierung besonders günstig sind, da sie Feldelemente adressieren, die mit konstantem Abstand im Speicher liegen. Darüber hinausgehende Vektorisierungen werden softwaremäßig realisiert und sind weniger schnell.

Beispiel:

```
      DO 40 K = 100,1,-1
            I = I + 1
40    A(K) = B(J*K + L,J) + C(I*K)
```

A und B sind in Vektorreferenzen verwendet, C nicht. Die Schleife wird vom CFT77 vektorisiert (wobei das Laden der Elemente von C eine *gather*-Funktion erfordert, da die Feldelemente nicht konstanten Abstand haben), vom CFT hingegen nicht.

Vektorreduktion (vector reduction)

Die Verknüpfung der Komponenten eines Vektors in einen Skalar durch eine arithmetische Operation in der Form

skalar = skalar +-*/ (vektorisierbarer Ausdruck)

erzeugt aus Vektoren eine skalare Größe. Dieser Vorgang wird als Vektorreduktion bezeichnet. Die skalare Reduktionsvariable darf in dem vektorisierbaren Ausdruck nicht auftreten. Vom CFT werden INTEGER-Reduktionsausdrücke mit Multiplikation oder Division nicht vektorisiert.

Beispiel:

```
      X = 0.0
      DO 50 I = 1,N
50    X = X + A(I)*B
```

Pseudovektor

Pseudovektor heißt eine Feldreferenz, die nicht die Kriterien für eine Vektorreferenz erfüllt, die aber einen vektorisierbaren Indexausdruck hat. Die Feldreferenz wird dann als skalare Unterschleife in der Vektorschleife behandelt, wobei der Indexausdruck als Vektorausdruck in der Vektorschleife berechnet wird.

Maschinen mit Hardware für einen sogenannten komprimierten Index (compressed index hardware, am ZIB nicht installiert) können die skalare Unterschleife mit einem Befehl berechnen.

12.4.3 Was vektorisieren CFT77 bzw. CFT nicht?

DO-Schleifen, die eines der nachfolgenden Elemente enthalten, werden derzeit nicht vektorisiert:

CFT *und* CFT77:

* innere Schleifen
* Abhängigkeitskonflikte in der Schleife
* Rückwärtssprünge innerhalb der Schleife
* Ein-/Ausgabeanweisungen
* Aufrufe von Unterprogrammen, sowie Funktionen für die keine Vektorversion vorliegt
* Variablen, Felder oder Funktionen vom Typ CHARACTER
* eine der Anweisungen RETURN, STOP oder PAUSE
* DEBUG-Option -eD (nicht aber mit Option -ez beim CFT77, siehe Kapitel 8.1.3)

Nur bei CFT:

* nichtlineare Feldindizes (z.B. indizierte Indizierung)
* ELSE-IF-Anweisungen
* geschachtelte IF-Anweisungen
* Überprüfung der Feldindizes auf Bereichsüberschreitung (Option -eo, siehe Kapitel 8.1.3)
* wenn die Schleife länger ist, als die für die Optimierung festgelegte maximale Blocklänge (Option -M, siehe Kapitel 8.3.2)

Datenabhängigkeiten und Abhängigkeitskonflikte

Vektorisierung bedeutet eine Veränderung der Ausführungsreihenfolge von Operationen. Welche Operationsreihenfolgen neben der im Programm vorgegebenen (mit "skalarer" Interpretation) erlaubt sind, ergibt sich aus den Datenabhängigkeiten. Verhindern die vorliegenden Datenabhängigkeiten eine Vektorisierung, liegt ein sogenannter *Abhängigkeitskonflikt* vor.

Beispiel a:

```
      DO 10 I=1,64
            A(I) = A(I) + B(I)
10    CONTINUE
```

Bei skalarer Ausführung werden die Vektorkomponenten einzeln geladen, addiert und abgespeichert. Bei vektorieller Bearbeitung werden die rechtsstehenden Vektoren aus dem Hauptspeicher in Vektorregister geladen und alle Vektorkomponenten aufaddiert, ehe eine Abspeicherung der linksstehenden Resultate beginnt. Diese Änderung der Reihenfolge hat keinen Einfluß auf die Resultate, daher darf die Schleife vektorisiert werden. Die äquivalente Vektoroperation lautet (in FORTRAN 8X-Schreibweise)

```
      A(1:N) = A(1:N) + B(1:N)
```

Beispiel b:

```
      DO 10 I=1,N
            A(I) = A(I-1) + B(I)
10    CONTINUE
```

Diese Schleife kann nicht in die Vektoroperation

```
      A(1:N) = A(0:N-1) + B(1:N)
```

übersetzt werden, da die linksstehenden Eingabedaten A(I-1) (abgesehen von A(0)) vor Beginn der Vektoraddition nicht in einem Vektorregister vorliegen können: die Ein- und Ausgabedaten der Anweisung überlappen sich, es liegt ein Abhängigkeitskonflikt vor.

Beispiel c:

```
      DO 10 I=1,N
            A(I) = C(I)
            B(I) = A(I+1)
10    CONTINUE
```

Bei sequentieller Verarbeitung werden dem Feld B die noch unveränderten Werte von A zugewiesen; bei naiver Vektorisierung

```
      A(1:N) = C(1:N)
      B(1:N) = A(2:N)
```

wäre dies nicht der Fall. Der CFT umgeht das Problem, indem er in einem Vektorregister einen temporären Vektor zum Zwischenspeichern von A(I+1) so anlegt, als hätte man geschrieben

```
      DO 10 I=1,N
            T = A(I+1)
            A(I) = C(I)
            B(I) = T
10    CONTINUE
```

Der CFT77 dagegen kann die Schleife nicht vektorisieren; er ist auf Zusammenarbeit mit dem FORTRAN-Präprozessor *fpp* angewiesen (siehe Kapitel 8.5). *fpp* analysiert die Datenabhängigkeiten und bemerkt, daß eine Vertauschung der beiden Anweisungen möglich ist und das Problem löst. Er erzeugt folgenden FORTRAN-Code:

```
CDIR$ IVDEP
      DO 10 I=1,N
            B(I) = A(I+1)
            A(I) = C(I)
10    CONTINUE
```

Beispiel d:

```
      DO 10 I=1,N
            A(I) = B(I-1) + C(I)
            B(I) = R * C(I)
10    CONTINUE
```

Zur Berechnung der Elemente A(I) werden ab dem zweiten Schleifendurchlauf die neu berechneten Elemente B(I) benötigt, d.h. die Eingabedaten der ersten Anweisung hängen von den Ausgabedaten der zweiten Anweisung ab. Eine direkte Übersetzung in Vektoroperationen

```
      A(1:N) = B(0:N-1) + C(1:N)
      B(1:N) = R * C(1:N)
```

würde zu falschen Resultaten führen. Die rückwärts gerichtete Datenabhängigkeit erlaubt wieder eine Vertauschung der Anweisungen. Die Compiler CFT und CFT77 erkennen dies nicht, aber *fpp* ist zu solchen (und auch weitaus komplizierteren) Programmtransformationen in der Lage.

Beispiel e:

```
      DO 10 I=1,N
            A(I) = B(I-1) + C(I)
            B(I) = R * A(I)
10    CONTINUE
```

Gegenüber dem vorigen Beispiel ist noch eine weitere Datenabhängigkeit hinzugekommen: das Eingabedatum A(I) der zweiten Anweisung hängt nun von der ersten Anweisung ab. Da auch die umgekehrte Abhängigkeit besteht, ist eine Vertauschung der Anweisungen nicht mehr möglich. Solche "Abhängigkeitskreise" verhindern generell die Vektorisierung.

12.4.4 Maßnahmen zur Unterstützung der Vektorisierung

Manche der hier zusammengestellten Hinweise und Ratschläge können veralten und hinfällig werden, da die Übersetzer ständig verbessert werden.

Verwendung des CFT77 (statt des CFT)

Der Autovektorisierer des CFT77 ist dem des CFT deutlich überlegen, wenn man von wenigen Ausnahmefällen absieht. Bei der Optimierung von Programmen sollte man sich jedoch darüber im klaren sein, daß die weiterreichenden Fähigkeiten des CFT77 oft durch Einsatz nur teilweise vektorisierter Software-Lösungen erkauft werden; die Vektorisierungsfähigkeiten des CFT spiegeln insofern die Menge der Sprachkonstrukte, die sich besonders effizient auf die Hardware abbilden lassen, deutlicher wieder. Gegen die Verwendung des CFT spricht weiterhin, daß Kompatibilität von *fpp* und CFT nicht gewährleistet ist.

Verwendung des *fpp*

Vektorisierung einer Schleife setzt voraus, daß drei Bedingungen erfüllt sind:

a) Zwischen den Anweisungen der Schleife dürfen keine Abhängigkeitskreise aus nicht-eliminierbaren Datenabhängigkeiten bestehen (vgl. Beispiel e: in 12.4.2), abgesehen von einfachen Vektorreduktionen und gewissen Formen linearer Rekursionen.
b) Nach Feststellung der Vektorisierbarkeit aufgrund der vorliegenden Datenabhängigkeiten muß der Übersetzer einen konstruktiven Weg zu einer Umstrukturierung der Schleife finden, so daß durch die Verwendung von Vektorbefehlen die Datenabhängigkeiten nicht verletzt werden.
c) Die in den Anweisungen der Schleife auftretenden Operationen und Funktionen müssen sich durch Vektorinstruktionen bzw. Vektorfunktionen ausdrücken lassen.

Bedingung c) ist auf CRAY-Rechnern, abgesehen von weniger typischen Anwendungsfällen wie etwa der Bearbeitung von Texten, kein Problem: alle elementaren und gängigen höheren Rechenoperationen wie auch die meisten FORTRAN-Standardfunktionen (intrinsic functions) stehen als einfache Vektoroperationen oder Vektorfunktionen zur Verfügung.

Mögliche Schwachpunkte bei den Übersetzern können somit nur die Datenabhängigkeitsanalyse a) oder die Programmtransformationen b) darstellen. Beide Funktionen sind im *fpp*, dem CFT77 und in rudimentärer Form auch dem CFT enthalten. Der *fpp* ist, hinsichtlich dieser Funktionen, dem CFT77 deutlich überlegen. FORTRAN-Programme sollten daher vom *fpp* vorübersetzt und dann vom CFT77 zu Objektcode übersetzt werden. Weiteres zur Verwendung von *fpp* in Kap 8.5 und DOC CRYFORT FPP. Eine Einführung in das Themengebiet "Datenabhängigkeiten, Vektorisierung und Parallelisierung" zusammen mit einer kompletten Beschreibung des *fpp* findet sich in dem Technical Report TR 90-4 des ZIB (siehe Anhang B.3).

Unterstützung der Abhängigkeitsanalyse

Die gründlichste Abhängigkeitsanalyse wird vom *fpp* vorgenommen. Aus verschiedenen Gründen ist eine vollständige Abhängigkeitsanalyse nicht immer möglich. Können die Datenabhängigkeiten zwischen zwei Anweisungen zur Übersetzungszeit nicht vollständig aufgeklärt werden, so bleiben zwei Möglichkeiten: entweder werden mehrere Code-Varianten für die verschiedenen im Prinzip denkbaren Abhängigkeitsstrukturen erzeugt, zusammen mit einem Laufzeittest, der zwischen den Alternativen auswählt, oder es wird sicherheitshalber Abhängigkeit angenommen - was unter Umständen eine Vektorisierung verhindert.

Beispiel a:

Aus dem Unterprogramm

```
      SUBROUTINE SHIFTH (A,N,M,K)
      REAL A(-N:N)
      DO 10 I=0,M
            A(I) = 0.5 * A(I+K)
10    CONTINUE
      RETURN
      END
```

erzeugt *fpp* ein Programm, das für die beiden möglichen Abhängigkeitsfälle Code und einen Laufzeittest enthält:

```
      SUBROUTINE SHIFTH (A,N,M,K)
      REAL A(-N:N)
      IF (ABS(K).GE.M+1 .OR. K.EQ.0) THEN
CDIR$ IVDEP
            DO 10 I = 1, M + 1
                  A(I-1) = 0.5*A(K+I-1)
10          CONTINUE
      ELSE
CDIR$ NOVECTOR
            DO 77001 I = 0, M
                  A(I) = 0.5*A(I+K)
77001       CONTINUE
CDIR$ VECTOR
      ENDIF
      RETURN
      END
```

Die Übersetzung dieser FORTRAN-Routine liefert den optimalen Code. In bedeutend komplizierteren Fällen kann jedoch Hilfestellung durch den Anwender erforderlich sein. Mit Hilfe von CFPP$-Direktiven kann der Anwender *fpp* Informationen zukommen lassen. Beispielsweise läßt sich mit

```
      CFPP$ RELATION (i1.rel.i2)
```

wobei *i1*, *i2* INTEGER-Grössen sind und *rel* einer der Vergleichsoperatoren GT, LT, GE, LE, EQ, NE (mit FORTRAN-Bedeutung), eine Relation etwa für Schleifengrenzen oder Inkremente in die Datenabhängigkeitsanalyse einbringen.

Ein Fall, wo die Übersetzer aufgrund unzureichender statischer Information sicherheitshalber Abhängigkeit annehmen, ist folgender:

Beispiel b:

```
      DO 10 I=1,N
            A(IND(I)) = A(IND(I)) + B(I)
10    CONTINUE
```

Da das Feld mit indizierter Adressierung auf beiden Seiten der Zuweisung auftritt, besteht die Möglichkeit, daß Feldelemente beim Durchlauf der Schleife mehrfach angesprochen werden. Dies verhindert eine Vektorisierung. Oftmals ist jedoch per Konstruktion sichergestellt, daß ein Wert im Indexfeld IND nicht mehrfach vorkommt, d.h. eine Permutation möglicher Indizes von A enthält. Dies festzustellen, überfordert heutige Übersetzer. Mittels einer Direktive kann der Anwender diese Information dem *fpp* mitteilen:

```
CFPP$ PERMUTATION (IND)
      ...
      DO 10 I=1,N
           A(IND(I)) = A(IND(I)) + B(I)
10    CONTINUE
```

Die Schleife wird nun von *fpp* als vektorisierbar deklariert.

Beispiel c:

Auch *fpp* liefert nicht in jedem Fall eine perfekte Abhängigkeitsanalyse, d.h. nimmt manchmal Datenabhängigkeiten an, die nicht vorhanden sind. Folgendes Beispiel der Symmetrisierung einer Matrix wird von keinem Übersetzer der CRAY als vektorisierbar erkannt:

```
      REAL A(N,N)

      DO 10 J=1,N
      DO 10 I=J,N
           A(I,J) = 0.5 * (A(I,J) + A(J,I))
10    CONTINUE
```

Nur wenn man die Diagonalelemente von der Symmetrisierung ausschließt, was unabhängig von der Vektorisierung sinnvoll ist, wird vektorisiert:

```
      REAL A(N,N)

      DO 10 J=1,N
      DO 10 I=J+1,N
           A(I,J) = 0.5 * (A(I,J) + A(J,I))
10    CONTINUE
```

Tatsächlich liegt aber auch im ersten Fall mit Einschluß der Diagonalelemente keine Datenabhängigkeit vor, die eine Vektorisierung verhindern könnte. Keiner der Übersetzer ist jedoch in der Lage, dies zu erkennen.

Beispiel d:

Wird eine komplexe Matrix hermitisiert,

```
      COMPLEX A(N,N)

      DO 10 J=1,N
      DO 10 I=J,N
           A(I,J) = 0.5 * ( A(I,J) + CONJG(A(J,I)) )
10    CONTINUE
```

so können die Diagonalelemente nicht mehr ausgenommen werden. Neben der Möglichkeit, die Diagonalelemente in einer separaten Schleife zu behandeln,

```
      COMPLEX A(N,N)

      DO 10 J=1,N
      DO 10 I=J+1,N
            A(I,J) = 0.5 * ( A(I,J) + CONJG(A(J,I)) )
10    CONTINUE
      DO 20 I=1,N
            A(I,I) = REAL(A(I,I))
20    CONTINUE
```

kann auch die Unabhängigkeit der einzelenen Schleifenoperationen durch Einfügen der Direktive CDIR$ IVDEP explizit konstatiert werden:

```
      COMPLEX A(N,N)

      DO 10 J=1,N
CDIR$ IVDEP
      DO 10 I=J,N
            A(I,J) = 0.5 * ( A(I,J) + CONJG(A(J,I)) )
10    CONTINUE
```

Vorsicht bei der Direktive IVDEP

Ist man sicher, daß bei der Ausführung keine Abhängigkeit auftritt, läßt sich die entsprechende Schleife mit der Compiler-Direktive IVDEP vektorisieren. Dies gilt auch dann, wenn eine Abhängigkeit aufgrund des Codes angenommen werden muß und sich die Unabhängigkeit erst aus den zur Laufzeit auftretenden Datenwerten ergibt. Aus zwei Gründen ist bei der Verwendung der Direktive Vorsicht geboten:

a) Da die Abhängigkeiten der Schleife von den Übersetzern nicht untersucht werden und unbesehen vektorisiert wird, führt das Auftreten echter Abhängigkeiten zu falschen Resultaten.

b) An der CRAY X-MP kann es wegen der Möglichkeit des gleichzeitigen Lesens und Schreibens vom und zum Hauptspeicher bei der Vektorverarbeitung mit einem einzigen Feld zu Überlappungen und damit ebenfalls zu falschen Ergebnissen kommen.

Das Einsetzen von IVDEP-Direktiven sollte im Regelfalle dem *fpp* überlassen werden. Nur wenn *fpp* trotz offensichtlicher Unabhängigkeit der Daten keine Vektorisierbarkeit anzeigt, sollte nach sehr sorgfältiger Prüfung der jeweiligen Situation die Direktive 'von Hand' eingesetzt werden. Ergebnisvergleiche werden dringend empfohlen.

IF-Anweisungen, Vektormischoperationen

Bedingte Anweisungen bereiten den neuen Compiler-Versionen beim Vektorisieren deutlich weniger Probleme als den früheren Versionen. Zwar kann der CFT 1.16 weder ELSE-IF-Anweisungen noch geschachtelte IF-Anweisungen vektorisieren, doch ist der CFT77 dazu in der Lage. Insbesondere erzeugen CFT und CFT77 für in Schleifen auftretende Anweisungen der Form

IF (var.op.ausdruck) var = ausdruck

wobei *op* ein Vergleichsoperator (GT, GE, LT, LE) ist, vektorisierten Code. Die früher zur Vektorisierung erforderliche Umwandlung in Vektorfunktionen

var = funktion(var,ausdruck)

mit *funktion* = MIN / MAX ist inzwischen nicht mehr unbedingt nötig. Haben *var* und *ausdruck* denselben Typ, ist auch *fpp* zu dieser Transformation in der Lage.

Beispiele:

```
IF (A(I) .GT. B(I)) A(I)=B(I)
```

wird zu

```
A(I) = AMIN1(B(I),A(I))
```

und

```
IF ( (I+3)*R1 .GT. R2 ) R2 = (I+3) * R1
```

wird zu

```
R2 = AMAX1((I+3)*R1,R2)
```

Der von den Compilern daraus erzeugte Objektcode ist fast gleich dem ohne Verwendung der Funktionen MIN / MAX erzeugten. Zumindest in diesen einfachen Fällen bringt die Transformation keinen Vorteil mehr. In komplizierteren Fällen könnte sie jedoch weiterhin vorteilhaft sein.

Die früher zur Vektorisierung der allgemeineren Form

```
IF (bedingung) var = ausdruck
```

notwendigen Aufrufe der Vektormerge-Funktion

```
var = CVMGT(ausdruck, var, bedingung)
```

sind auch nicht mehr zwingend erforderlich. Der für die beiden Codevarianten von den Compilern erzeugte Code unterscheidet sich sehr, die Transformation lohnt sich in manchen Fällen immer noch. Bei Verwendung der Vektormerge-Funktionen wird der Ausdruck auf der rechten Seite der Zuweisung zunächst für alle Vektorelemente berechnet; lediglich die Zuweisung zur Zielvariablen wird in Abhängigkeit von der IF-Bedingung durch das Vektormaskenregister gesteuert. Diese Vorgehensweise kann folgende Nachteile haben:

* Ist die Bedingung nur in wenigen Fällen erfüllt, so kann dies zu erheblichem Mehraufwand führen.
* Enthält der Ausdruck auf der rechten Seite Divisionen oder Aufrufe von Intrinsic-Funktionen, so kann die vektorielle Verarbeitung zu Laufzeitfehlern (z.B. wegen einer Division durch Null) führen.

inline-Expansion

FORTRAN-Unterprogramme können *inline* expandiert, das heißt, Aufrufe von Unterprogrammen können durch die Unterprogramme selbst ersetzt werden. *inline*-Expansion bringt durch Wegfall von Unterprogrammaufrufen Laufzeit-Verbesserungen im Prozentbereich. Werden durch Expansion jedoch Schleifen vektorisierbar, sind darüber hinaus große Leistungssteigerungen möglich.

Beispiel:

Das *cf77*-Übersetzersystem bietet zwei Möglichkeiten zur *inline*-Expansion:

* Expansion durch den CFT77 mit der Option

-I inline_file,

wobei *inline_file* eine Datei ist, die die zu expandierenden Unterprogramme enthält.

* Expansion durch den *fpp* mit den Optionen

-e und den Schaltern 6, 7, 8 mit folgender Bedeutung:

6	*sichere* Expansion von Unterprogrammen, die gewissen Einfachheitskriterien genügen
7	*aggressive*, möglicherweise fehlerhafte Expansion von Unterprogrammen, die abgeschwächten Kriterien genügen
8	Suchen der zu expandierenden Routinen nicht in gleichnamigen Dateien, z.B. Routine *x* oder *X* in der Datei *x.f* (was der Voreinstellung entspricht), sondern in der Eingabedatei, die auch die Aufrufe enthält.
-I r1,r2, ...	wobei *r1, r2,* ... Unterprogramme sind, die *inline* expandiert werden sollen.
-M lines	wobei *lines* die maximale Anzahl von Programmzeilen derjenigen Unterprogramme angibt, die zu expandieren sind.
-S file1, file2 ...	zur Angabe von Dateien, die zu expandierende Unterprogramme enthalten.

Die CFPP$-Direktiven AUTOEXPAND, EXPAND und SEARCH erlauben eine lokale Steuerung der *inline*-Expansion. Eine ausführliche Beschreibung der Optionen und Direktiven von *fpp* sowie praktische Hinweise und eine Besprechung möglicher Fehlerquellen bei der *inline*-Expansion findet sich in DOC CRYFORT FPP.

Die *inline*-Expansion erfolgt beim CFT77 in einem Zwischencode, so daß der expandierte Code für den Anwender (außer im Assembler) nicht sichtbar ist. Beim *fpp* hingegen geschieht die Expansion in FORTRAN, kann also nachvollzogen werden. Wir empfehlen *inline*-Expansion durch den *fpp* nicht "blind" zu verwenden; beachten Sie auf jeden Fall die Anmerkungen in den Abschnitten 4.6 und 6 des DOC CRYFORT FPP.

Beispiel: *inline*-Expansion durch *fpp*

Aus dem Codefragment

```
      parameter (n=10000)
      real a(n),b(n),d(n)
      ...
      kmin=1
      kmax=N/2
      do 10 i=1,n
CFPP$ EXPAND
            call subr(n,a,irange(i,kmin,kmax),b,i,c,d)
10    continue
      ...
      end
```

mit den Unterprogrammen

```
      integer function irange (i,kmin,kmax)
      if (i.lt.kmin) then
      irange=kmin
      else if (i.gt.kmax) then
            irange=kmax
      else
            irange=i
      endif
      return
      end

      subroutine subr (n,a,ind,b,i,c,d)
      real a(n),b(n),d(n)
      a(ind) = b(i) * c
      d(i)   = a(ind) + 1.0
      return
      end
```

erzeugt *fpp* mit der Option *-e68* den Code (modifizierte bzw. geänderte Zeilen in Großschreibung):

```
      parameter (n=10000)
      real a(n),b(n),d(n)
      INTEGER IRAN1X
      ...
      kmin=1
      kmax=N/2
      do 10 i=1,n
CFPP$ EXPAND
      IF (I .LT. 1) THEN
            IRAN1X = 1
      ELSE IF (I .GT. KMAX) THEN
            IRAN1X = KMAX
      ELSE
            IRAN1X = I
      ENDIF
      A(IRAN1X) = B(I)*C
      D(I) = A(IRAN1X) + 1.0
10    continue
      ...
      end
```

Die beiden letzten Anweisungen der Schleife erfordern GATHER- und SCATTER-Befehle, so daß auf der CRAY-Anlage des ZIB keine vollständige Vektorisierung möglich ist; die Schleife wird nach Expansion vom CFT77 (teil-)vektorisiert. Wählt man für *fpp* zusätzlich die Option *-dc* (keine Parallelisierung - eine solche hat ohnehin nicht stattgefunden), wird folgender FORTRAN-Code erzeugt, der noch mehr auf die Vektorisierung zugeschnitten ist:

```
      parameter (n=10000)
      INTEGER J1S,IRAN1U(10000)
      REAL R1V(10000)
      LOGICAL L1S,L2S
      REAL QQQ
      COMMON/Z1FPP0CM/ QQQ(65535)
      EQUIVALENCE (QQQ(1),IRAN1U),(QQQ(10001),R1V)
      real a(n),b(n),d(n)
      INTEGER IRAN1X
      kmin=1
      kmax=N/2
      ...
CDIR$ IVDEP
      DO 10 I = 1, 10000
      L1S = .FALSE.
      L2S = .FALSE.
      IF (I .LT. 1) THEN
            IRAN1U(I) = 1
      ELSE
            L2S = I .GT. KMAX
            L1S = .NOT.L2S
      ENDIF
      IF (L2S) IRAN1U(I) = KMAX
      IF (L1S) IRAN1U(I) = I
      R1V(I) = B(I)*C
10    CONTINUE
      DO 77001 I = 1, 10000
            A(IRAN1U(I)) = R1V(I)
            D(I) = A(IRAN1U(I)) + 1.0
77001 CONTINUE
      ...
      end
```

Beide Schleifen werden nun vom CFT77 vektorisiert.

12.5 Weitere Optimierungsmöglichkeiten an der CRAY

12.5.1 Vergrößerung der Vektorlänge

Die Verarbeitungsgeschwindigkeit bei Vektorverarbeitung hängt u.a. wesentlich von der Länge der Vektoren (Anzahl der Schleifendurchläufe) ab. Je länger die Vektoren sind, desto höher ist die Verarbeitungsgeschwindigkeit pro durchgeführter Operation. Ist die Vektorlänge kleiner als 4, so lohnt sich eine Vektorisierung nicht. Da nur innere DO Schleifen vektorisiert werden, sollten die längsten Vektoren in den inneren Schleifen verarbeitet werden. Oft läßt sich die Vektorlänge auch dadurch vergrößern, daß geschachtelte Schleifen auf einfache Schleifen reduziert werden, indem mehrdimensionale Felder auf eindimensionale Felder abgebildet werden.

Beispiel: nach Melenk, H.: Strategie bei der Multiplikation vieler kleiner Matrizen mit Hilfe von CRAY-Vektorrechnern, ZIB, Interner Bericht 1984:

Hundert kleine Matrizen (3,3) seien miteinander zu multiplizieren.

```
      DO 300 M = 1,100
         DO 230 K = 1,3
            DO 200 J = 1,3
               C(M,K,J) = 0.0
200         CONTINUE
            DO 220 I = 1,3
               DO 210 J = 1,3
                  C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210            CONTINUE
220         CONTINUE
230      CONTINUE
300   CONTINUE
```

Hier werden die Matrixprodukte einzeln nacheinander berechnet. Die innere Schleife 210 vektorisiert mit der unzureichenden Vektorlänge von 3.

Hinweis: Wegen des automatischen Abrollens von Schleifen bis zur Vektorlänge 9 wird auch Schleife 220 von CFT vektorisiert. Bei CFT77 gibt es diese Leistung nicht. Ein besseres Laufzeitverhalten erhält man durch:

```
         DO 140 K = 1,3
            DO 130 J = 1,3
               DO 120 M = 1,100
                  C(M,K,J) = 0.0
120            CONTINUE
130         CONTINUE
140      CONTINUE
         DO 300 K = 1,3
            DO 230 J  = 1,3
               DO 220 I  = 1,3
                  DO 210 M = 1,100
                     C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210               CONTINUE
220            CONTINUE
230         CONTINUE
300      CONTINUE
```

In der inneren Schleife werden jetzt hundert Matrizen simultan mit der Vektorlänge 100 berechnet.

Da *fpp* Schleifen vertauschen kann, läßt sich die längste Schleife auch mittels *fpp* nach innen legen. Für die erste Codevariante erzeugt *fpp*:

```
      DO 300 K = 1, 3
            DO 77001 J = 1, 3
CDIR$ IVDEP
                  DO 77002 M = 1, 100
                        C(M,K,J) = 0.0
77002             CONTINUE
77001       CONTINUE
      DO 77003 I = 1, 3
            DO 77004 J = 1, 3
CDIR$ IVDEP
                  DO 77005 M = 1, 100
                        C(M,K,J) = C(M,K,J) + A(M,K,I)*B(M,I,J)
77005             CONTINUE
77004       CONTINUE
77003 CONTINUE
300   CONTINUE
      RETURN
      END
```

Bei Übersetzung der zweiten Code-Variante bemerkt *fpp* zusätzlich, daß sich die Schleifen 120, 130 und 140 zu einer zusammenfassen lassen und generiert

```
CDIR$ IVDEP
      DO 140 K = 1, 900
            C(K,1,1) = 0.0
140   CONTINUE
            DO 300 K = 1,3
                  DO 230 J  = 1,3
                        DO 220 I  = 1,3
CDIR$ IVDEP
                              DO 210 M = 1,100
                                    C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210                           CONTINUE
220                     CONTINUE
230               CONTINUE
300         CONTINUE
      RETURN
      END
```

Diese Version hat nur den Nachteil, daß in Schleife 140 bei eingeschalteter Feldgrenzenkontrolle zur Laufzeit Fehler gemeldet werden. Eine bessere Lösung ist daher die Einführung eines zusätzlichen eindimensionalen Feldes, das mittels EQUIVALENCE über das Feld C gelegt wird:

```
      REAL A(100,3,3), B(100,3,3), C(100,3,3), C1(100*3*3)
      EQUIVALENCE (C,C1)
      ...
      DO 120 MKJ = 1,3*3*100
            C1(MKJ) = 0.0
120   CONTINUE
            DO 300 K = 1,3
                  DO 230 J  = 1,3
                        DO 220 I  = 1,3
                              DO 210 M = 1,100
                                    C(M,K,J) = C(M,K,J) + A(M,K,I) * B(M,I,J)
210                           CONTINUE
220                     CONTINUE
230               CONTINUE
300         CONTINUE
      RETURN
      END
```

Generell sollte versucht werden, die Dimensionierung der Felder (Reihenfolge der Indizes) so einzurichten, daß

* der in der zeitaufwendigsten, vektorisierbaren Schleife veränderliche Index an erster Stelle im Feld steht (Vermeidung von Bankkonflikten, s.u.);
* sich Laufindizes geschachtelter Schleifen zu jeweils einem Index zusammen fassen lassen (eventuell auch unterschiedliche Kombinationen an verschiedenen Stellen des Programms).

12.5.2 Abrollen von Schleifen (unrolling)

Schleifen *abrollen* bedeutet, Schleifen zu beseitigen oder die Zahl der Schleifendurchläufe dadurch zu vermindern, daß die zu wiederholenden Operationen ausdrücklich hingeschrieben werden.

Vorteilhaft ist es, wenn durch das Abrollen in der innersten Schleife mehr Operationen und kompliziertere arithmetische Ausdücke entstehen. Der Übersetzer kann dann die Register und die Verkettung von Operationen besser ausnutzen. Ein weiterer Vorteil kann durch die Beseitigung kurzer innerer Schleifen entstehen, wenn dadurch eine Schleife mit größerer Vektorlänge zur innersten wird.

Innere Schleifen mit einer konstanten kleinen Zahl von Durchläufen kann schon der Übersetzer abrollen (siehe Kapitel 8.3.2, Option -u, nur CFT!). Das folgende Beispiel zeigt, wie durch die Verminderung von Durchläufen einer äußeren Schleife die innere Schleife mit Vektoroperationen angereichert wird. Dabei sorgt die Klammerung für die beste Ausnutzung der Register.

Beispiel:

nach Dongarra, J.J.; Eisenstat. S.C.: Squeezing the most out of an Algorithm in CRAY-FORTRAN, ACM TOMS, Vol. 10 No.3 1984, S. 219-230:

```
      SUBROUTINE SMXPY4 (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM,*)
C
C   PURPOSE:
C   Multiply matrix M times vector X and add the result to
C   vector Y.
C
C   PARAMETERS:
C   N1   INTEGER, number of elements in vector Y, and number of
C        rows in matrix M
C   Y    REAL(N1), vector of length N1 to which is added the
C        product M*X
C   N2   INTEGER, number of elements in vector X, and number of
C        columns in matrix M
C   LDM  INTEGER, leading dimension of array M
C   X    REAL(N2), vector of length N2
C   M    REAL(LDM,N2), matrix of N1 rows and N2 columns
C
C   Cleanup odd vector
C
     J = MOD(N2,2)
     IF (J .GE. 1) THEN
        DO 10 I = 1, N1
           Y(I) = (Y(I)) + X(J)*M(I,J)
 10     CONTINUE
     ENDIF
C
C    Cleanup odd group of two vectors
C
     J = MOD(N2,4)
     IF (J .GE. 2) THEN
        DO 20 I = 1, N1
           Y(I) = ( (Y(I))
  $               + X(J-1)*M(I,J-1)) + X(J)*M(I,J)
 20     CONTINUE
     ENDIF
C
C   Main loop - groups of four vectors
C
    JMIN = J+4
    DO 40 J = JMIN, N2, 4
       DO 30 I = 1, N1
          Y(I) = ((( (Y(I))
   $               + X(J-3)*M(I,J-3)) + X(J-2)*M(I,J-2))
   $               + X(J-1)*M(I,J-1)) + X(J)  *M(I,J)
30     CONTINUE
40  CONTINUE
C
    RETURN
    END
```

SMXPY4 nutzt die Register und die Verkettung von Operationen besser als die folgende Version:

```
      SUBROUTINE SMXPY (N1, Y, N2, LDM, X, M)
      REAL Y(*), X(*), M(LDM,*)
C
       DO 20 J = 1, N2
          DO 10 I = 1, N1
             Y(I) = (Y(I)) + X(J)*M(I,J)
 10       CONTINUE
 20    CONTINUE
C
       RETURN
       END
```

12.5.3 Operationen ohne Hardware-Unterstützung

INTEGER-Multiplikationen und -Divisionen vermeiden

INTEGER-Multiplikationen und -Divisionen sind auf der CRAY erheblich langsamer als die entsprechenden REAL-Operationen (Multiplikation: bis Faktor 7, Division: bis Faktor 20). Deshalb sollten in diesen Fällen möglichst REAL-Arithmetik oder 46-Bit INTEGER benutzt werden.

Die Übersetzer-Option *-o fastmd* (nur CFT, siehe Kapitel 8.3.2) bewirkt, daß für ganzzahlige Multiplikation die Gleitkommaeinheit verwendet wird. Die Zahlen sind damit durch die Mantissenlänge auf 2**46 beschränkt (siehe Kapitel 12.1).

In Einzelfällen kann auch die Verwendung von INTEGER*2 (24-Bit INTEGER) sinnvoll sein. Wenn ganze Zahlen zu verarbeiten sind, die sicher kleiner als 2**24 bleiben, können auch die Funktionseinheiten des Adreßteils verwendet werden (siehe Kapitel 12.1 und 8.1.5/8.2.4 für CFT77 bzw. 8.4.4 für CFT).

DOUBLE PRECISION vermeiden

Für DOUBLE PRECISION-Operationen stehen auf der CRAY keine Hardware-Instruktionen zur Verfügung. Daraus ergeben sich erhebliche Verlangsamungen bei der Ausführung gegenüber normaler REAL-Arithmetik (Addition, Subtraktion und Division: bis Faktor 60; Multiplikation: bis Faktor 35).

DOUBLE PRECISION-Operationen sollten deshalb nur verwendet werden, wenn einfache Genauigkeit nicht ausreicht (doppelte Genauigkeit der CRAY entspricht ungefähr der vierfachen Genauigkeit einer IBM oder CONVEX).

Mit Hilfe der Option *-ep* (siehe Kapitel 8.1.4) kann ein Programm, das in doppelter Genauigkeit geschrieben wurde, mit der einfach genauen Arithmetik übersetzt werden.

12.6 Optimierung von Ein-/Ausgabeoperationen und Speicherzugriffen

Wegen der sehr begrenzten Hauptspeicherkapazität (32 MByte - 4 MW) der CRAY des ZIB muß bei vielen Anwendungen auf die vergleichsweise langsamen Hintergrundspeicher (Plattenspeicher, SDS) zurückgegriffen werden. Bei manchen Anwendungen ist die Optimierung von Ein-/Ausgabeoperationen daher wichtiger, als die Optimierung der Rechenoperationen. Eine erschöpfende Behandlung des Themas ist an dieser Stelle nicht möglich, nur die wichtigsten Punkte können angesprochen werden. Für weitergehende Informationen sei verwiesen auf DOC CRYFORT FORTIO und DOC CRAY SSD.

Die Techniken der I/O-Optimierung sind:

* Minimierung von Zugriffen auf Hintergrundspeicher
* Benutzung der schnelleren Hintergrundspeicher in der Speicherhierarchie
* Verwendung schneller I/O-Methoden (unformatiert, ...)
* Überlappung der Ein- und Ausgabevorgänge mit Rechenoperationen

12.6.1 Minimierung der Anzahl von Hintergrundspeicherzugriffen

Zugriffe auf Hintergrundspeicher sollten möglichst vermieden werden und ersetzt werden durch Zugriffe auf den Hauptspeicher. Dies läßt sich erreichen durch

* Ersetzung mehrerer Einzelzugriffe auf den Hintergrundspeicher durch einen Blockzugriff auf den Hintergrundspeicher und Einzelzugriffe auf den Hauptspeicher
* speichersparende Programmierung (z.B. durch Mehrfachverwendung von Feldern), so daß eine umfangreichere Datenhaltung im Hauptspeicher möglich wird
* Verwendung speichersparender Algorithmen

12.6.2 Verwendung schneller Hintergrundspeicher

Am langsamsten sind I/O-Operationen über NFS (Network Filesystem), auf Platten anderer Rechner (Plattenbereich $PERM auf der CRAY X-MP des ZIB). Dieser Plattenspeicher sollte nur zur längerfristigen Aufbewahrung von Dateien zwischen einzelnen Läufen verwendet werden. Zur Bearbeitung sollten solche Dateien in jedem Falle auf CRAY-Platten kopiert werden, die um Größenordnungen schneller sind. Zur Beschleunigung von I/O-Operationen auf den CRAY-Platten ist die CRAY X-MP des ZIB mit einem Erweiterungsspeicher (SSD, 128 MW) in Halbleitertechnologie ausgerüstet (siehe Kapitel 2.7). Er ist über einen sehr schnellen Kanal (1,2 GByte/s) mit dem Hauptspeicher verbunden. Der Datentransfer über diese Leitung läuft asynchron, d.h. die CPU arbeitet während des Tansfers weiter. Der größte Teil des SSD dient als schneller Pufferspeicher (*ldcache*, 105 MW) zwischen Hauptspeicher und dem Plattenbereich */tmp*. Alle I/O-Operationen auf dem Plattenbereich */tmp* werden automatisch gepuffert. Diese Pufferung ist besonders wirksam, wenn aufeinanderfolgende Zugriffe auf den Hintergrundspeicher möglichst dicht beieinander liegen. Der Performance-Gewinn hängt ab von der Trefferrate und der I/O-Methode; er kann bis zu einem Faktor von 100 reichen. Im SSD steht ein weiterer Bereich als Ergänzung des Hauptspeichers (SDS, 11 MW) Anwendungsprogrammen auf Anforderung zur Verfügung. Dieser Erweiterungsspeicher wird vom Programm aus wie eine Datei auf einer Platte angesprochen, jedoch wird der SSD mehr als eine Erweiterung des Hauptspeichers als als eine schnelle Platte angesehen. Ab UNICOS 6.0 ist die Möglichkeit vorgesehen, Felder von COMMON-Blöcken im SDS abzulegen und vom Programm aus wie andere Felder anzusprechen. Zur Verwendung des SDS siehe DOC CRAY SSD.

Die wichtigsten Punkte bei der Auswahl von Hintergrundspeicher sind somit:

* keine I/O-Operationen über NFS
* bei Ein- und Ausgabe auf CRAY-Platten den durch den *ldcache* gepufferten Plattenbereich */tmp* wählen
* Effizienz der Pufferung sichern durch gute Kohärenz der Daten (geeignete Programmierung und insbesondere Verwendung geeigneter Algorithmen)
* wenn möglich SDS nutzen:
    - er bietet schnellstes I/O
    - bringt keine Belastung der CPU
    - braucht fast keine I/O-Wait-Time (da die Daten sicher vorhanden sind und nicht von der Platte nachgeladen werden müssen

12.6.3 Verwendung schneller und asynchroner I/O-Methoden

Unabhängig davon, welches Speichermedium angesprochen wird, weisen die verschiedenen I/O-Methoden sehr unterschiedliches Zeitverhalten auf. Im Standard-FORTRAN sind beispielsweise formatierte (zum Lesen und Schreiben "lesbarer" Textinformation) und unformatierte (Ein- und Ausgabe von Binärinformation) I/O-Operationen vorgesehen, sowohl mit sequentiellem als auch direktem Zugriff. Neben diesen Standardmethoden bietet CRAY, wie viele andere Hersteller, spezielle, auf die Rechnerarchitektur zugeschnittene Erweiterungen an (s. DOC CRAYFORT FORTIO).

Der wichtigste Punkt ist die Verwendung unformatierter Ein- und Ausgabe: diese kann bis zu 2000 mal schneller sein als formatierte! Beim Lesen und Schreiben größerer Informationsmengen sollte nur die wirklich vom Menschen zu lesende Information in formatierter Form erzeugt werden.

Ein weitere Methode, I/O-Operationen zu beschleunigen, gibt es auf der CRAY durch die Möglichkeit, ungepuffert zu schreiben. Normalerweise werden, da Daten auf Platten oder in den SSD nur in Blöcken von 512 Worten geschrieben und gelesen werden, bei allen I/O-Vorgängen die Daten in einem Puffer gesammelt, bis er voll ist. Erst dann findet der eigentliche I/O-Vorgang statt. Dieses *Library-Buffering* kann der FORTRAN-Benutzer ausschalten, wenn er in seinem Programm selbst dafür sorgt, daß die Ein- und Ausgabe in ganzzahligen Vielfachen von Blöcken zu 512 Worten erfolgt.

Weiterhin gibt es auf der CRAY die Möglichkeit, I/O-Operationen asynchron durchzuführen. Normalerweise werden die I/O-Operationen synchron abgewickelt, d.h. nach Anstoß einer I/O-Operation wartet die CPU mit der Ausführung der nächsten Instruktion des Benutzerprogrammes, bis die I/O-Operation beendet ist. Bei asynchronen I/O-Operationen hingegen arbeitet die CPU sofort weiter.

Für Informationen zu den Kombinationsmöglichkeiten, der Performance und der praktischen Realisierung der verschiedenen I/O-Methoden siehe in DOC CRAYFORT FORTIO und den FORTRAN-Reference-Manuals.

12.6.4 Minimierung von Speicherzugriffskonflikten

Da die Speicherzellen im Hauptspeicher der CRAY nach einem Zugriff für einige Taktzyklen nicht ansprechbar sind, ist der gesamte Hauptspeicher in vier Abschnitte zu je acht sogenannten "Bänken" aufgeteilt. Jedes Rechenwerk hat drei Pfade (Ports) zum Hauptspeicher, ein weiterer Pfad pro Rechenwerk führt vom Hauptspeicher zum I/O-Subsytem.

Aufgrund dieser Konstruktion gibt es Speicherzugriffskonflikte (siehe 12.1.6), die dazu führen, daß Speicherzugriffe unterschiedlich lange dauern. Die Wechselwirkung der Speicher-Zugriffsströme der CRAY X-MP ist im allgemeinen sehr komplex. Sie hängt ab von den Strategien zur Konfliktbehebung, dem Typ des Zugriffs (skalar oder vektoriell), der Startadresse des Vektors (legt den ersten Pfad und die erste Bank fest), dem Abstand der Vektorelemente im Speicher, den Vektorlängen und der Zahl aktiver Ports und CPUs. Trotz dieses komplexen Geschehens gibt es sehr einfache Regeln zur Minimierung von Bank-belegt-Konflikten. Der Rechner ist so angelegt, daß maximal ein Wort pro Verbindung und Takt übertragen werden kann. Im Hauptspeicher aufeinanderfolgende Worte sind auf die 32 Bänke verteilt. Wird ein Vektor gelesen oder geschrieben, dessen Elemente den Abstand eins haben (in FORTRAN: Elemente eines Feldes, die sich im ersten Index um eins unterscheiden), so dauert es 32 Zyklen, bis wieder auf dieselbe Bank zugegriffen wird. Problematisch wird es, wenn mit jedem (oder jedem zweiten, dritten oder vierten) Takt auf dieselbe Speicherbank zugegriffen wird: dann entstehen Bank-belegt-Konflikte.

Beispiel (Vermeidung von Bankkonflikten):

```
      REAL A(1024,20), B(1024,20)

      DO 10 J = 1, 20
            DO 10 I = 1,1024,32
                  A(I,J) = C * B(I,J)
10    CONINUE
```

Da in Schleife 10 jedes 32. Feldelement von A und B angesprochen wird, gibt es (falls durch andere Aktivitäten in der Maschine sonst keine Verzögerungen eintreten) mit jedem Speicherzugriff einen Bankkonflikt. Das gleiche ist der Fall, wenn im zweiten Index vektorisiert wird:

```
      REAL A(1024,20), B(1024,20)

      DO 10 I = 1,1024
            DO 10 J = 1, 20
                  A(I,J) = C * B(I,J)
10    CONINUE
```

Hier haben nacheinander angesprochene Elemente im Hauptspeicher den Abstand 1024, ein Vielfaches von 32. Durch einen einfachen Trick, nämlich geringfügige Überdimensionierung, die nur etwas Speicherplatz kostet, läßt sich das Problem jedoch beheben:

```
      REAL A(1025,20), B(1025,20)

      DO 10 I = 1,1024
            DO 10 J = 1, 20
                  A(I,J) = C * B(I,J)
10    CONINUE
```

Die nacheinander angesprochenen Feldelemente haben nun Abstand 1025, das heißt zeitlich aufeinander folgende Bänke haben Abstand 1, so daß erst nach 32 Zugriffen wieder dieselbe Bank aktiv werden muß. Aus dieser Betrachtung ergeben sich folgende Regeln:

* wenn im ersten Index (in FORTRAN: dem linksstehenden Index) von Feldern vektorisiert wird, sollte das Schleifeninkrement kein Vielfaches von 32, 16, 8 oder 4 sein
* wird in einem höheren Index vektorisiert, so ist die Gefahr von Bankkonflikten besonders groß: der Abstand der nacheinander angesprochenen Speicherzellen ergibt sich aus dem Produkt der niedrigeren Dimensionslängen und dem Schleifeninkrement; dieses Produkt ist bei den üblichen Dimensionierungen sehr häufig ein Vielfaches von 4, 8, 16 oder gar 32.
* durch Überdimensionierung der niedrigeren Dimensionen (in FORTRAN: der links des in der Schleife laufenden Index stehenden Dimensionen) kann man sicherstellen, daß nacheinander angesprochene Feldelemente in unterschiedlichen Bänken liegen.

12.7 Benutzung optimierter CRAY Routinen - die Bibliothek SCILIB

Einige Situationen in FORTRAN-Programmen lassen eine Vektorisierung auch nach Modifikation des FORTRAN-Quellprogramms nicht zu oder die volle Leistung kann erst in Assembler erreicht werden. Insbesondere in diesen Fällen ist die Verwendung hochoptimierter CRAY-Bibliotheksroutinen sinnvoll.

Diese Routinen ermöglichen in einigen Fällen eine Vektorisierung auch da, wo in FORTRAN 77 eine Vektorisierung nicht möglich wäre (z.B. Minimum-/Maximumsuche in einem Vektor); in anderen Fällen erlauben sie zumindest eine effizientere (nicht-vektorisierende) Bearbeitung von Problemen, als sie in FORTRAN 77 möglich wäre (z.B. Auflösen von linearen Rekursionen). Unter Umständen empfiehlt sich die Benutzung von anderen CRAY-optimierten Programmen, z.B. aus der linearen Algebra (Berechnung des Skalarprodukts, Matrizenprodukte, Lösung linearer Gleichungssysteme etc.).

In diesem Abschnitt wird ein Überblick über Unterprogramme gegeben, die nicht zum FORTRAN 77 Standard gehören. Sie sind von CRAY zur Verfügung gestellt, um bestimmte Aufgaben in einer dem Rechner und seiner Architektur angepaßten Weise lösen zu können. Eine vollständige und ausführliche Dokumentation findet man im UNICOS Math and Scientific Library Reference Manual SR -2081 (siehe Anhang B.1).

Diese Routinen stehen ohne besondere Maßnahmen jedem Benutzerprogramm zur Verfügung, da sie in Systembibliotheken residieren, die beim Laden automatisch benutzt werden. Benutzerprogramme mit gleichen Namen haben beim Laden Vorrang, da der Lader die Systembibliotheken standardmäßig als letzte durchsucht. Werden die Systemroutinen gewünscht, obwohl in anderen beim Laden angegebenen Bibliotheken Routinen gleichen Namens vorhanden sind, so ist durch eine geeignete Reihenfolge sicherzustellen, daß die Systemroutinen verwendet werden (siehe Kapitel 4.3). Die in diesem Abschnitt angegebenen Routinen residieren in der Bibliothek */lib/libsci.a* (Kurzform für *segldr*, Option *-l: sci*).

Die folgende Übersicht enthält die wichtigsten Routinen. Es ist zu beachten, daß bei neueren Betriebssystemversionen neue Routinen hinzukommen können. Andererseits verlieren unter Umständen einige Routinen durch verbesserte Fähigkeiten der Übersetzer CFT77 bzw. CFT ihre Berechtigung.

Teilweise sind die Aufrufe mit Parametern angegeben. Dabei bedeuten:

N	Vektorlänge
SX, SY	Vektoren vom Typ REAL
INCX, INCY	Inkremente für SX bzw. SY (Für aufeinanderfolgende Elemente =1)
SA, R	REAL Variable
I	INTEGER Variable

Grundoperationen der linearen Algebra

Die standardisierte Sammlung von Routinen für Grundoperationen der linearen Algebra, genannt Basic Linear Algebra Subprograms (BLAS), ist Bestandteil der SCILIB. Die Sammlung besteht aus drei Paketen (level-1-BLAS, level-2-BLAS und level-3-BLAS), die folgende Routinen enthalten:

BLAS 1: grundlegende Vektoroperationen
BLAS 2: Matrix-Vektoroperationen
BLAS 3: Matrix-Matrix-Operationen

Alle BLAS-Routinen sind als CAL-Routinen für einfach genaue und komplexe Arithmetik in der SCILIB verfügbar. Sie lassen sich von FORTRAN, C und CAL aufrufen.

BLAS 1 Routinen sind zum Beispiel:

SASUM	Summe der Absolutbeträge eines Vektors. Aufruf: SASUM(N,SX,INCX)
SAXPY	Multiplikation einer Konstanten mit einem Vektor und Addition eines anderen Vektors (Triade). Y = a * X + Y. Aufruf: CALL SAXPY(N,SA,SX,INCX,SY,INCY)

SDOT	Skalarprodukt zweier Vektoren. Aufruf: SDOT(N,SX,INCX,SY,INCY)
SNRM2	Euklidische Norm eines Vektors. Aufruf: SNRM2(N,SX,INCX)
SSUM	Summe der Elemente eines Vektors. Aufruf: SSUM(N,SX,INCX)

Bei der Optimierung ist zu beachten, daß es inzwischen oftmals günstiger ist, diese Funktionen auszuschreiben, da die neuen Compiler-Versionen ebenso effizienten Code erzeugen; neben dem Vorteil, daß der Unterprogrammaufruf entfällt, wird durch die Berücksichtigung des Kontextes eine bessere Registerbelegung möglich.

Lineare Rekursionen

FOLR FOLRN	Auflösung linearer Rekursionen erster Ordnung vom Typ C(I) = - A(I)*C(I-1) + B(I). Kann zur Berechnung des Hornerschemas benutzt werden.
SOLR SOLRN	Auflösung linearer Rekursionen zweiter Ordnung vom Typ C(I+2) = A(I)*C(I+1) + B(I)*C(I)
SOLR3	Auflösung linearer Rekursionen zweiter Ordnung mit drei Gliedern vom Typ C(I) = C(I) + A(I-2)*C(I-1) + B(I-2)*C(I-2).

Ausgeschriebene lineare Rekursionen werden in bestimmten Fällen vom *fpp* erkannt und in Aufrufe der entsprechenden SCILIB-Routinen verwandelt.

Weitere numerische Algorithmen

Die mit *) bezeichneten Routinen sind neben der Dokumentation im UNICOS Math and Scientific Library Reference Manual zusätzlich durch 'Technical Notes', herausgegeben von der Firma CRAY, erläutert.

MINV*)	Lösung eines linearen Gleichungssystems mit mehreren rechten Seiten. Berechnung der Inversen und der Determinante einer quadratischen, regulären Matrix.
MXM*) MXMA*)	Produkt Matrix * Matrix
MXV MXVA	Produkt Matrix*Vektor
FILTERG*) FILTERS*) OPFILT*)	Berechnungen für lineare Filterverfahren
CFFT2*) RCFFT2*) CRFFT2*)	Fourier Transformationen (Analyse und Synthese) für reelle und komplexe Argumente

GATHER und SCATTER

Wenn Vektoren zu verarbeiten sind, die nicht gleichabständig im Speicher liegen, sondern über ein Indexfeld indiziert werden, helfen diese optimierten Routinen beim Umspeichern. Das lohnt sich bei großen Vektorlängen und/oder, wenn der umgespeicherte Vektor mehrfach in einem vektorisierbaren Ausdruck verwendet wird.

Hardware zur Unterstützung von GATHER und SCATTER ist an der CRAY X-MP des ZIB nicht installiert. Daher sollten Vektoren möglichst immer gleichabständig im Speicher liegen.

GATHER — Sammeln von Vektorelementen mit indizierter Indizierung. Diese Routine ersetzt FORTRAN-Code der Form

```
      DO 1 I = 1,N
1        A(I) = B(INDEX(I))
```

SCATTER — Verteilen von Vektorelementen mit indizierter Indizierung. Diese Routine ersetzt FORTRAN-Code der Form

```
      DO 1 I = 1,N
1        A(INDEX(I)) = B(I)
```

Indizierte Indizierung verhindert beim CFT die Vektorisierung, so daß Umspeichern in gleichabständige Vektoren, Rechnen mit diesen und eventuell Verteilen der Resultate meist deutlich günstiger ist. Dazu sind die beiden Routinen gedacht. Der CFT77 erzeugt selbst Code für die Umspeicherung, die eine Vektorverarbeitung ermöglicht, d.h. für Schleifen mit indizierter Indizierung wird teilvektorisierter Code erzeugt (in der Übersetzerliste werden solche Schleifen als "vektorisiert" ausgewiesen).

Such- und Sortierroutinen

ISMIN, ISMAX — Indexbestimmung des kleinsten bzw. größten Vektorelements.
Aufruf: ISMxx (N,SX,INCX)

ISAMIN, ISAMAX — Indexbestimmung des betragsmäßig kleinsten bzw. größten Vektorelements.
Aufruf: ISAMxx (N,SX,INCX)

IILZ — Anzahl der Werte =0 (oder = .FALSE.) vor dem ersten Wert ungleich Null

ILLZ — Anzahl der Werte > =0 (oder = .FALSE.) vor dem ersten Wert < 0

ILSUM — Anzahl der Werte <0 (oder = .TRUE.) in einem Vektor.

ISRCHxx — Suchen der Position des ersten Elements eines Vektors, das zu einem vorgegebenen Wert in der Beziehung .EQ., .NE., .LT., .LE., .GT. oder .GE. steht.

OSRCHx — Suchen der Position des ersten Elements eines geordneten Vektors, das einen gegebenen Wert hat.

CLUSxx — Suchen der Positionen von Teilvektoren, deren Elemente zu einem vorgegebenen Wert in der Beziehung .EQ., .NE., .LT., .LE., .GT. oder .GE. stehen.

WHENxx — Suchen der Anzahl und der Positionen aller Elemente eines Vektors, die zu einem vorgegebenen Wert in der Beziehung .EQ., .NE., .LT., .LE., .GT. oder .GE. stehen.

ORDERS Bestimmen der Sortierindizes für Records fester Länge, die in einem zweidimensionalen Feld gespeichert sind.

LINPACK

LINPACK ist ein Paket zur Analyse und Lösung linearer Gleichungssysteme. Die Dokumentation befindet sich in folgendem Buch (auch als CRAY-Publikation S1-0113 erhältlich):

LINPACK User's Guide
J.J. Dongarra, J.R. Bunch, C.B. Moler, G.W. Stewart
SIAM, 1979
ISBN 0-89871-172-X

Folgende Matrizenklassen (bzw. Zerlegungen) werden behandelt (die beiden angegebenen Buchstaben stehen an 2. und 3. Stelle im Routinennamen):

GE	General
GB	General band
PO	Positive definite
PP	Positive definite packed
PB	Positive definite band
SI	Symmetric indefinite
SP	Symmetric indefinite packed
TR	Triangular
GT	General Tridiagonal
PT	Positive definite tridiagonal
CH	Cholesky decomposition
QR	Orthogonal triangular decomposition
SV	Singular value decomposition

Von dem Unterprogrammpaket LINPACK stehen alle einfach genauen Routinen in einer für CRAY optimierten Form zur Verfügung (1. Buchstabe der Routinennamen ist S). Die Versionen mit komplexer Arithmetik sind auch in der SCILIB vorhanden, allerdings in weniger optimierter Form.

EISPACK

Von dem Unterprogrammpaket EISPACK stehen die Routinen für reelle Matrizen zur Verfügung, die das Eigenwertproblem

$$Ax = gx$$

lösen. Diese Routinen sind für die CRAY-Anlage optimiert. EISPACK ist dokumentiert in:

B.T. Smith, J.M. Boyle et al.
Matrix Eigensystem Routines -Eispack Guide
Volume 6, second edition

und

B.S. Garbow, J.M. Boyle et al.
Matrix Eigensystem Routines -Eispack Guide extension
Volume 51

Beide Bände sind im Springer Verlag in der Reihe "Lecture Notes in Computer Science" erschienen.

Folgende Matrizenklassen werden behandelt:

General
Symmetric
Symmetric band
Symmetric packed
Symmetric tridiagonal
Special tridiagonal
Upper Hessenberg

Sowohl die LINPACK- wie auch die EISPACK-Routinen haben denselben Namen, denselben Algorithmus und dieselbe Aufruffolge wie die Originalversionen. Zur Optimierung wurden folgende Schritte vorgenommen:

* Entfernung von IF-Anweisungen, bei denen das Resultat beider Zweige gleich ist;

nur bei LINPACK:

* Ersetzung der Aufrufe der BLAS-Routinen SSCAL, SSWAP, SAXPY und SROT durch *inline*-FORTRAN-Code, der von den FORTRAN-Compilern vektorisiert wird;
* Ersetzung von SDOT bei der Lösung linearer Dreiecks-Systeme durch besser vektorisierbaren Code;

nur bei EISPACK:

* Verwendung von BLAS-Routinen SDOT, SASUM, SNRM2, ISAMAX und ISMIN, wo möglich;
* Abrollen von Schleifen;
* Verwendung von CDIR$ IVDEP, wo keine die Vektorisierung verhindernden Datenabhängigkeiten bestehen.

Die Optimierungen bei LINPACK sind somit mehr an den Fähigkeiten der neuen FORTRAN-Compiler-Versionen orientiert als die bei EISPACK.
Die Routinen beider Pakete sollten die gleichen Resultate wie die Originalversionen liefern. Bei numerisch instabilen Verhältnissen kann es auf Grund der die Operationsreihenfolge verändernden Optimierungen jedoch zu deutlichen Abweichungen kommen.

Anhang A. Vom ZIB bereitgestellte Programmpakete

Anwendungssoftware aus dem wissenschaftlichen Bereich ist für CRAY-Rechner in großem Umfang verfügbar. Der Grad der Anpassung an die speziellen Vektorrechnereigenschaften ist dabei unterschiedlich. Über die von CRAY selbst beziehbare Anwendungssoftware gibt der "Application Software Library Catalog" einen Überblick. Eine umfangreiche Aufstellung über weitere, auch von dritter Seite angebotene Software enthält das "Directory of UNICOS Applications Software for CRAY Supercomputer". Beide Schriften liegen in den Rechenzentren zur Einsicht bereit. Interessenten an bestimmten Anwendungspaketen sollten sich an die Beratung ihres Rechenzentrums oder direkt an das Konrad-Zuse-Zentrum (ZIB) wenden.

Folgende Software ist an der CRAY verfügbar (in Klammern der ACM-Index):

ADINA (T4)

ADINA (Automatic Dynamic Incremental Nonlinear Analysis) ist ein Programmsystem zur statischen und dynamischen Analyse der Verschiebungen und Spannungen von Festkörpern und Strukturen sowie zur Analyse von Flüssigkeiten. Die derzeit an der CRAY bereitgestellten Versionen von ADINA sind nur für Benutzer der Technischen Universität Berlin und der Bundesanstalt für Materialprüfung verfügbar. Sofern auch von anderen Benutzern Interesse an der Nutzung von ADINA besteht, muß geprüft werden, ob eine Bereitstellung für alle Benutzer des ZIB durchgeführt werden kann.

BIZEPS2 (J)

Oberhalb von GKS angesiedelte Programmbibliothek für allgemeine grafische Anwendungen.
Weitere Information: DOC,BIZEPS2

BLAS (F1)

Basic Linear Algebra Subprograms: CRAY-Assembler Routinen, hochoptimiert für die Nutzung auf der CRAY. BLAS ist Bestandteil von = > SCILIB*) sowie = > LINPACK.
Literatur:
UNICOS Math and Scientific Library Reference Manual, SR-2081 5.0, Volume 3

EISPACK (F2)

EISPACK ist ein Unterprogrammpaket für FORTRAN zur Berechnung von Eigenwerten und Eigenvektoren von verschiedenen Klassen von Matrizen (z.B. reell symmetrisch, komplex, Hessenberg-Form, usw.). Auf der CRAY ist EISPACK Bestandteil von = > SCILIB*).
Literatur:
- Smith et al., Matrix Eigensystem Routines-EISPACK Guide, Lecture Notes in Computer Science, Springer-Verlag, 1974
- H. Wilkinson, C. Reinsch: Handbook for Automatic Computation Volume 2, Linear Algebra, Springer-Verlag 1971
UNICOS Math and Scientific Library Reference Manual, SR-2081 5.0, Volume 3

GAUSSIAN 86 (T2)

Programmsystem für "ab initio" Berechnungen in der Quantenchemie, entwickelt an der Carnegie-Mellon-University.
Weitere Information: DOC,CRAY,GAUS86

GRIPS (J5)

Die Unterprogrammbibliothek GRIPS (Grafik für interaktive und passive Systeme) kann auf der CRAY bei allgemeinen grafischen Anwendungen von FORTRAN-Programmen aufgerufen werden. GRIPS basiert auf dem Graphischen Kernsystem GKS. (An der CRAY-Anlage ist z.Zt. nur die passive Variante GRIPS1 verfügbar.)
Weitere Information: DOC,GRIPS bzw. DOC,CRAY,GRIPS.

IMSL (V0)

Die Programmbibliothek IMSL besteht aus FORTRAN-Unterprogrammen aus verschiedenen Gebieten der Mathematik und Statistik.
Weitere Information: DOC,IMSL und DOC,CRAY,IMSL

LINPACK (F4)

LINPACK ist ein Unterprogrammpaket für die Lösung und Analyse von Linearen Gleichungssystemen. Auf der CRAY ist LINPACK Bestandteil von = > SCILIB*).
Literatur:
- J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart:
LINPACK User's Guide; Philadelphia 1979:
Society for Industrial and Applied Mathematics (SIAM)

LISP

List Processor ist eine höhere Programmiersprache mit dynamischer Speicherverwaltung zur Entwicklung von Programmen für die Manipulation hochstrukturierter symbolischer Daten. LISP ist als Portable Standard Lisp (PSL) in der Version 3.4 an der CRAY X-MP des ZIB implementiert.
Literatur:
H.Melenk/W.Neun: Portable Standard LISP for CRAY X-MP Computers, SC 86-2, Dezember 1986, ZIB.

NAG (V0)

NAG ist eine umfangreiche Programmbibliothek, bestehend aus FORTRAN-Unterprogrammen.
Weitere Information: DOC,NAG und DOC,CRAY,NAG

REDUCE

Die auf LISP aufbauende Formelmanipulationssprache REDUCE steht in der Version 3.3 zur Verfügung.
Literatur:
H. Melenk/W. Neun: REDUCE User's Guide for the CRAY-1/CRAY X-MP Series running UNICOS Version 3.3. ZIB Technical Report 88-2.
DOC,CRAY,REDUCE

SCILIB*) (V0)

Die SCILIB (Scientific Applications Subprograms) ist eine von der Firma CRAY Research entwickelte bzw. angepaßte Programmbibliothek; die Routinen sind für die Nutzung auf der CRAY hochoptimiert und in allen bekannten Fällen entsprechenden Routinen anderer Bibliotheken überlegen. SCILIB enthält Routinen aus folgenden Gebieten:

lineare Algebra (BLAS level 1, 2 und 3)
funktionale und lineare Rekursion
LINPACK
EISPACK
Matrix-Inversion und -Multiplikation
Gather und Scatter
Fast Fourier Transformation
Filter-Routinen
Such- und Sortierroutinen

Literatur:
UNICOS Math and Scientific Library Reference Manual, SR-2081 5.0, Volume 3

An der NOS/BE-Anlage von TU und ZIB existiert darüber hinaus eine weitgehend aufrufkompatible Nachbildung wichtiger Teile der von CRAY bereitgestellten Bibliothek SCILIB für CD-Anlagen, die insbesondere für die Programmentwicklung eingesetzt werden kann. Weitere Information: DOC,SCI.

*) Die unter UNICOS verfügbaren Routinen der Bibliothek SCILIB sind in der Datei *-lib/libsci.a* enthalten. Im *segldr*-Kommando wird diese Bibliothek automatisch durchsucht.

eLib

Von der Gruppe *Software Information* des Konrad-Zuse-Zentrum für Informationstechnik Berlin wird eine *elektronische Software Library - eLib* aufgebaut, die den Zugriff auf mathematische Software des ZIB, die Recherche nach Algorithmen für bestimmte Problemstellungen und den Durchgriff auf die *netlib* (Sammlung von ca. 5000 Routinene von AT&T) ermöglicht.

Erste Informationen sind erhältlich durch Absenden des Kommandos *help* per *email*:

X.400: A = dbp, P = ZIB-Berlin, OU = sc, S = elib

RFC822: elib@sc.ZIB-Berlin.dbp.de

oder derzeit auch über Dialog unter der WiN-Adresse 45 050 331 033 mit *userid* = *elib* und *password* = *elib*.

Anhang B. Literatur

B.1 Literatur der Fa. CRAY Research, Inc. (Stand April 1990)

Bestell-Nummer	Titel	Preis in DM (zuzgl. 7% MwSt)
Betriebssystem UNICOS		
SMN-7000 6/89	UNICOS Philosophy Übersicht über UNICOS	1,35
SG-2010 C	UNICOS Primer Einführende Informationen über UNICOS mit Beispielen	58,50
SG-2052 A	UNICOS Overview for Users Nützliche Einführungsinformation für UNICOS-Erstbenutzer	35,00
SQ-2056 5.0	User Commands Ready Reference Kompakt-Broschüre über UNICOS-Kommandos und Shell-Merkmale	20,60
SR-2011 5.0	UNICOS User Commands Reference Manual beschreibt sämtliche UNICOS-Kommandos in alphabetischer Reihenfolge der Kommandonamen	136,35
SR-2012 5.0	Volume 4: UNICOS System Calls Reference Manual	136,35
SR-2048 5.0	UNICOS Index for CRAY-2 Computer Systems Besonders nützlich in Verbindung mit den UNICOS *man-pages*	49,85
Sprachen		
SG-0115	CFT Optimization Guide Optimierungstechniken in CFT für Fortgeschrittene	111,10
SN-0312 A	CFT77 Messages Auflistung aller Meldungen des CFT77-Compilers	6,10
SR-0009 M	Fortran (CFT) Reference Manual Beschreibt den CRAY FORTRAN Compiler und damit zusammenhängende Betriebssystem-Funktionen	131,10
SR-0018 C	CFT77 Reference Manual vollständige Beschreibung des CFT77-Compilers	100,85
SR-0060 E	Pascal Reference Manual	91,00
SR-0112 C	UNICOS Symbolic Debugging Package Reference Manual	42,50
SR-2024 D	Cray C Reference Manual Beschreibung der Programmiersprache C auf der CRAY	76,85
SR-2074 1.1	Cray Standard C Programmer's Reference Manual	62,90

Bibliotheken und Utilities

SN-2088 A	Autotasking User's Guide technische Beschreibung von Autotasking	108,75
SR-0013 J	UPDATE Reference Manual	63,75
SR-0066 F	Segment Loader (SEGLDR) and ld Reference Manual Funktionsbeschreibung des CRAY-Laders für segmentierte und unsegmentierte Programme	42,50
SR-0112 C	UNICOS Symbolic Debugging Package Reference Manual Beschreibung der DEBUG-Programme SYMDUMP, DRD und DDA	42,50
SR-0136 5.0	Cray C Library Reference Manual Beschreibung der C-Unterprogramme für die CRAYs	67,25
SR-0222 F	CRAY Y-MP, CRAY X-MP EA, and CRAY X-MP Multitasking Programmer's Manual Beschreibung von Multitasking und Microtasking	79,50
SR-2012 5.0	Volume 3: UNICOS System Calls Library Reference Manual	49,85
SR-2040 B	UNICOS Performance Utilities Reference Manual UNICOS-Hilfsprogramme für Struktur und Performance Analyse	36,25
SR-2079	Volume 1: UNICOS Fortran Library Reference Manual	91,00
SR-2080 C	Volume 2: UNICOS Standard C Library Reference Manual	50,00
SR-2081 5.0	Volume 3: UNICOS Math and Scientific Library Reference Manual	86,00

Stations und Netzwerkzugang

SC-0270 B	CDC NOS/VE Link Software Command Reference Manual Beschreibt die Leistungen der Kopplungssoftware zwischen CRAY und dem Vorrechner CD CYBER 930	68,00
SG 2009 D	TCP/IP Network User's Guide	34,60
SR-0034 F	CDC NOS/BE Station Reference Manual Beschreibt die Leistungen der Kopplungssoftware zwischen CRAY und dem Vorrechner CD CYBER 825	94,25

Technical Notes

SN-0210 A	Linear Digital Filters for CFT Usage Benutzerprogramme für lineare digitale Filter für CFT	2,50
SN-0206 B	Complex to Real Fast Fourier Transform Binary Radix Subroutine (CRFFT2)	2,50

SN-0216 A Subroutines for Packing und Unpacking 64-bit-Words 2,50
(PACK/UNPACK)

Die oben aufgeführte Originalliteratur dürfte in Ihrem Rechenzentrum vorhanden sein. Wir empfehlen, zunächst dort oder im ZIB das betreffende Manual anzusehen, wenn Sie Fragen haben, die über das in diesem Handbuch Ausgeführte hinausgehen. Sie können die Schriften auch direkt von

CRAY RESEARCH GMBH
Geschäftsbereich Software
z.H. Frau Claudia Mayer
Kistlerhofstr. 168
8000 München 70
Telefon: (089) 78590-0 (Direktwahl (089) 78590-131)

beziehen. Für die Richtigkeit der Preise können wir keine Gewähr übernehmen. Bei der Original-CRAY-Literatur wird zusätzlich zu den angegebenen Preisen die Mehrwertsteuer erhoben.

B.2 Literatur

zur CRAY:

Kay A. Robbins/Steven Robbins
The CRAY X-MP/Model 24
A Case Study in Pipelined Architecture and Vector Processing
ISBN 3-540-97089-4
Springer-Verlag 1989 Berlin Heidelberg New York

zu UNIX:

Wie in UNIX-Systemen üblich, sind auch in UNICOS wesentliche Teile der Beschreibung über das Kommando *man* online verfügbar (sog. *man-pages*). Das Kommando schreibt nach *stdout*, durch Ausgabeumlenkung also auch in eine beliebige Datei. Eine kurze Einführung zu *man* findet man in Kapitel 4.9.

Da UNIX ein auf sehr vielen unterschiedlichen Rechnern eingesetztes Betriebssystem ist, erhält man auch im normalen Buchhandel gute Fachinformationen zu UNIX. Stellvertretend für die große Palette von UNIX-Büchern sei hier aufgeführt (Bestellung über den Buchhandel!):

Gulbins, Jürgen
UNIX
Eine Einführung in Begriffe und Kommandos von UNIX-Version 7, bis System V.3
Dritte, überarbeitete und erweiterte Auflage
Springer-Verlag 1988 Berlin Heidelberg New York
ISBN 3-540-19248-4 3. Auflage. Gebunden 84.- DM

zu FORTRAN:

Wehnes, Harald
FORTRAN 77
5., überarbeitete Auflage 1989. Kartoniert 26.- DM
Carl Hanser Verlag München Wien
ISBN 3-446-15775-1

B.3 Handbücher der Rechenzentren

Deutschsprachige Literatur zu den Vorrechnern und zu FORTRAN:

Hege, H.C.
Datenabhängigkeitsanalyse und Programmtransformation auf
CRAY-Rechnern mit dem FORTRAN-Präprozessor fpp
ZIB Technical Report TR 90-4. Kostenlos
Konrad-Zuse-Zentrum für Informationstechnik Berlin

FORTRAN 77 Sprachumfang
unter dem CDC-Betriebssystem NOS/VE
Ein Nachschlagewerk
11., veränderte Auflage Juli 1987. 6,00 DM
RRZN-Klassifikationsschlüssel SPR.F77 1
RRZN, Universität Hannover

Franziska Städtler
Arbeiten mit NOS/VE
Eine Anleitung
5.4.1989. 13,00 DM
Fachhochschule Nürnberg

H.Hoffmann, D.Kehl, H.-J.Mildner, A.Preusser, R.Riedel, S.Schwenkler
NOS/BE Benutzeranleitung für das Betriebssystem
4. aktualisierte und korrigierte Auflage 1986. 8,00 DM
ZRZ der TU Berlin und ZIB

INTERCOM Instant
6. überarb. Auflage 1987, 2,00 DM
ZRZ der TU Berlin.
(Beschreibung aller wichtigen Kommandos im Dialogbetrieb,
insbes. FAMILY, TUBE, PAGE)

Die oben aufgeführten Handbücher sind zumeist in den Rechenzentren erhältlich. Bestellungen können auch an die

Verwaltung des
Konrad-Zuse-Zentrum für
Informationstechnik Berlin (ZIB)
Heilbronner Straße 10
1000 Berlin 31

gerichtet werden.

Unabhängig davon kann Einblick in die Originalhandbücher der Fa. Control Data ("CD-Manuals") gewährt werden; diese stehen in den Beratungsräumen der Rechenzentren zur Verfügung; z.T. können diese Handbücher auch ausgeliehen werden.

B.4 ZIB-Dokumentationssystem DOC

Weitere, insbesondere aktuelle Informationen erhält man auf beiden Vorrechnern der CRAY über das Online-Informationssystem DOC. Die Sachgebiete CRAY und CRYFORT stehen darüberhinaus über DOC abrufbar auch an den CD-Rechnern in FUB und TUB und an den SUN-Rechnern im ZIB zur Verfügung.

DOC kann sowohl im Dialog als auch im Stapelbetrieb aufgerufen werden mit

DOC,cat,item,L = list,MO = mode,PS = ps,PL = pl,REV = rev.

Parameter:

cat	Category: Sachgebiet, über das der Aufrufende informiert werden möchte.
item	Item: Begriff aus dem o.g. Sachgebiet, über den der Aufrufende informiert werden möchte. Fehlt die Angabe, so erhält er die zum Sachgebiet gehörende Schrift GENERAL, die einen Überblick über das Sachgebiet liefert, oder aber die Schrift INDEX mit einem Inhaltsverzeichnis dieses Sachgebietes. Mehrere Begriffe können durch "/" getrennt in einem Aufruf von DOC angefordert werden (nur NOS/BE).
= *	Information über alle DOCs des Sachgebietes (nur NOS/BE).
= all	Information über alle DOCs des Sachgebietes (nur NOS/VE).
L = list	Angabe des Namens der Datei, in die die DOC-Ausgabe erfolgen soll (nur NOS/BE).
= *	Ausgabe auf dem Zentraldrucker. Ohne Angabe dieses Parameters "*" erscheint die gewünschte Schrift bei Aufruf im Dialog auf dem Bildschirm, im Stapelbetrieb in der Ausgabeliste.
MO = mode MO(mode)	mode: Ausgabe von TITLE Titelfeld SHORT Kurzfassung LONG ausführlicher Dokumentation
PL = pl = 50	Print Limit: Anzahl der auszudruckenden DOC-Seiten, falls > 50
REV = rev	Ausgabe nur der DOCs, die **nach** dem Stichtag *tt/mm/jj* verändert wurden.

Sachgebiet CRAY: Einige für die Benutzung der CRAY wichtige Dokumente

ALRJE	Alternative Remote Job Entry auf der CRAY
BENCHLIB	Unterprogramme aus interner CRAY Bibliothek
DATEIORG	Filesysteme an der CRAY
DELJOB	Löschen eines Jobs auf der CRAY
ERRORS	Fehler in UNICOS und im ZIB Handbuch
GAUS86	Erste Hinweise zum Einsatz von GAUSSIAN 86
GKS	Nutzung von GKS auf der CRAY
GRIPS	Grips an der CRAY
HEADER	Aktueller Header der CRAY
IMSL	Besonderheiten der IMSL Installation an der CRAY unter UNICOS
LOGIN	Zugang zur CRAY unter UNICOS
NAG13	Überblick über die FORTRAN-Bibl. NAG ** MARK 13 **

NEWS	Aktuelle Informationen zum UNICOS-Betrieb
PASSWORD	Ablauf und Änderung des CRAY-Paßwortes
PRIORITY	Prioritätenregelung an der CRAY
REDUCE	REDUCE 3.3 für Cray
RST	Informationen über CRAY-Batchjobs
SSD	Nutzung des SSD an der CRAY X-MP des ZIB
TOOLS	Nützliche Hilfsmittel beim Arbeiten mit UNICOS
UNICOS51	Unterschiede UNICOS 5.1 zu UNICOS 4.0
VORJAHR	Nutzung der CRAY im abgelaufenen Jahr

Sachgebiet CRYFORT: Gibt Informationen zu CRAY FORTRAN Compilern CFT77 und CFT

CFT116	Compilerversion CFT 1.16: Änderungen gegenüber CFT 1.15
CFT77	Neuer Compiler CFT77
CFT77V3	CFT77 V 3.0
FORTIO	FORTRAN-Ein-/Ausgabe
FPP	Beschreibung des FORTRAN-Preprozessors FPP
HINWEISE	Allgemeine Hinweise zu den CRAY FORTRAN Compilern
NEWS	Aktuelles zu CFT und CFT77

Sachgebiet CRUTIL: Hilfsprogramme an der CRAY.

wo2be	Dateitransfer zwischen der CRAY des ZIB und FTP-fähigen Rechnern innerhalb des TU-Netzes WOTAN

Sachgebiet DFUE: Zugang zum ZIB über Netze und für Dialoggeräte.

TELEFON	DFUE - Zugang zu den Rechnern des ZIB für Dialoggeräte: Telefonanschlüsse

Sachgebiet BERNET: Einbindung der CYBER 825 (NOS/BE) in BERNET und DFN, Dienste: Remote Job Entry, Filetransfer, passiver Dialog, aktiver Dialog.

ALLGEMEIN	Beschreibung des BERNET-Verbundsystems
CDC	Benutzung von CDC-Anlagen über BERNET
RECHNER	Am BERNET-Verbundsystem beteiligte Rechner

Beschreibungen der BERNET-Dienste:

RJE	Remote Job Entry (Jobtransfer mit Hilfe von BERNET)
FT	Der File Transfer-Dienst
FTINSI	Beispiele für den File-Transfer-Dienst zwischen CD unter NOS/BE und VAX unter PCS-MUNIX
FTVMS	Beispiele für den File-Transfer-Dienst zwischen CD unter NOS/BE und VAX unter VMS
DIALOG	Der Dialog-Dienst auf CDC-Anlagen
SOFTPAD	Der CDC-Software-PAD

Sachgebiet DFN: Einbindung der CYBER 930 (NOS/VE) in DFN; Dienste: Remote Job Entry, Filetransfer, passiver Dialog.

FTVE	Filetransfer an der CYBER 930 unter NOS/VE
RJEVE	Remote Job Entry an der CYBER 930 unter NOS/VE
RJEVEBE	Beispiele für Remote Job Entry zwischen der CYBER 930 unter NOS/VE und der CYBER 825 unter NOS/BE
RJECRY	Beispiele für den Zugang zur CRAY X-MP über DFN-RJE

Sachgebiet ORGANISA: Allgemeine Informationen für die Anlagen des ZIB, u.a:

ADRESSEN	Anschriften der Partnerrechenzentren
BETRIEB	Übersicht über den Bereich Anlagenbetrieb des ZIB
DATENSCH	Datenschutzregelungen
ENTGELT	Entgeltordnung für die Rechenanlagen des ZIB
FREMDMT	Organisatorische Regelungen für Fremdbänder

Mit dem Kommando: DOC,ORGANISATION,*,SEL=BENORD. (unter NOS/BE) erhalten Sie die Kurzfassung aller Informationen, die Teil der betrieblichen Regelungen des ZIB sind.

Sachgebiet SPRACHEN: Übersicht über die am ZIB zur Verfügung stehenden Programmiersprachen.

Sachgebiet SYSTEM (NOS/VE) bzw. VESYSTEM (NOS/BE): Informationen über das NOS/VE-System im ZIB.

BAENDER	Magnetbandverarbeitung an der CYBER 930 (NOS/VE-Anlage)
BENUTZER	Organisatorische Vereinbarung über Benutzereinträge
CD930	Allgemeines zur CYBER 930
COMMANDS	Kommando - Kurzübersicht
ENVIRON	Bereitstellen von speziellen Kommando-Umgebungen
HEADER	Aktuelle Mitteilungen des ZIB (CYBER 825)
LOGIN	Login-Anleitung für VE im ZIB
NEWS	Neuigkeiten und Hinweise zur CYBER 930 unter NOS/VE
REDO	Kommando-Wiederholung
RST	Informationen über Jobs an der CRAY
VECRY	Zugang zur CRAY von NOS/VE aus

Sachgebiet SYSTEM (NOS/BE) bzw. BESYSTEM (NOS/VE): Beschreibung einer Reihe von zusätzlichen Programmen auf der CYBER 825.

ANATAPE	Analyse von Magnetbändern
BAENDER	Regelungen zur Benutzung von Magnetbändern
CANCEL	Herausnahme von Jobs aus der Input-Queue und Abbruch von aktiven Jobs durch den Benutzer
CLEARAC	Löschen fremder IDs aus der Erlaubnisliste
COMPACT	Komprimieren von Datenfiles
DOC	Allgemeines Dokumentationsprogramm
FAMILY	Family Indirect File System (Konzept und Kommandos)
FEXPORT	Erzeugen eines Jobs, der Textdateien (Quellen, Daten, Dokumente) satzorientiert im EBCDIC- oder ASCII-Code so auf ein Magnetband schreibt, daß dieses auch von anderen Betriebssystemen verarbeitet werden kann
FIMPORT	Erzeugen eines Jobs, der Textdateien (Quelle, Daten, Dokumente) satzorientiert im

	EBCDIC- oder ASCII-Code von einem Magnetband mit fester Recordlänge und Blockung liest, wie es von vielen Betriebssystemen geschrieben werden kann.
GIVEAC	Einräumen von Zugriffsrechten an fremde IDs
LFTRANS	Local-File-Transfer
LIMITS	Richtlinien für die Vergabe von Limits
LISTID	Information über eigene permanente Files
LISTPF	Liste der permanenten Files (Ersatz für AUDIT)
MBO	Anfordern von Magnetbändern im ZIB
MICRO	File-Transfer zwischen Cyber unter NOS/BE und PC
MTT	Senden einer Meldung von einem Job an ein Terminal
PASSWORTE	Paßworte im Batchbetrieb
PERMFILES	Regelungen zur Benutzung permanenter Dateien
PRINT	Allgemeines Druckkommando
PRO	Page Remote Output (Sichten von Batchjob-Listings)
QUEUES	Informationen über Jobs am eigenen Rechner
RST	Informationen über Jobs an anderen Rechnern
STACK	Manager für Jobumgebung
STARTUP	Startup-Prozedur beim Terminal-LOGIN
STATIS	Informationen über Betriebsmittel-Inanspruchnahme
STATUS	Informationen über Jobs in der CYBER 170-825
SYSTEM	Organisatorische Aspekte des Betriebssystems NOS/BE
TUBE	Zeileneditor

Sachgebiet KIEL: Besonderheiten zur Nutzung der CRAY X-MP der Universität Kiel.

CRAY	Benutzung der CRAY X-MP in Kiel
INFO	Organisatorisches zur Kieler CRAY X-MP
VMS	VMS an der VAX in Kiel

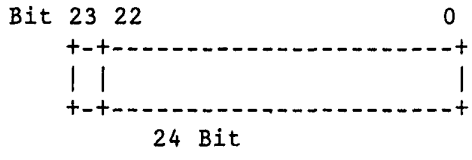
Anhang C. Zahlenbereiche, -Genauigkeit und -Speicherung

An der CRAY X-MP hat jedes Hauptspeicherwort 64 binäre Ziffern (Bit). Die Register haben entweder 64 Bit (V, T und S) oder 24 Bits (A und B). Darin werden Daten gespeichert. Die Mehrzahl der Programme verarbeitet die gespeicherten Daten als ganze Zahlen (INTEGER), Gleitkommazahlen (REAL), Buchstaben (CHARACTER) oder logische Werte (BOOLEAN). Alle Typen haben eine andere Darstellung im Hauptspeicher oder in den Registern. Die folgenden Kapitel zeigen die interne Darstellung der Typen INTEGER, REAL und BOOLEAN auf der CRAY; die Darstellung von Zeichen (CHARACTER) wird im Anhang D behandelt.

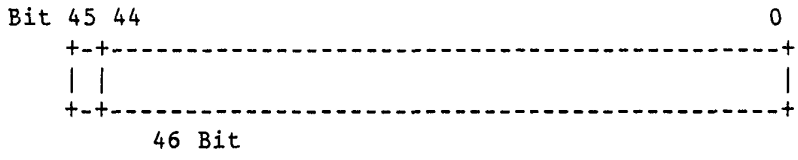
C.1 Ganze Zahlen (INTEGER)

Ganze Zahlen (24 Bit, 46 Bit oder 64 Bit) werden im Zweierkomplement verarbeitet. Dabei werden positive ganze Zahlen durch ihr binäres Äquivalent dargestellt. Eine negative ganze Zahl erhält man durch Komplementieren der absoluten Zahl und Addition von Eins. Daraus ergibt sich, daß das am weitesten links stehende Bit (Bit 63) den Wert Null für positive und Eins für negative Zahlen hat. Dieses Bit wird deshalb Vorzeichenbit genannt. Die Darstellung von ganzen Zahlen:

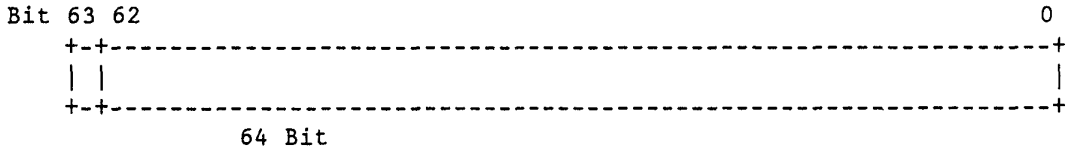
1. 24 Bit Integer:



2. 46 Bit Integer:



3. 64 Bit Integer:



Das folgende Beispiel zeigt die Darstellung der Zahl -29 in 24 Bit:

```
+29         ----->  000 000 000 000 000 000 011 101
Komplement  ----->  111 111 111 111 111 111 100 010
Addiere 1   ----->  111 111 111 111 111 111 100 011 = -29
```

Die Additions- und Multiplikationseinheiten für die Adreßrechnung verarbeiten nur 24-Bit Integerzahlen. Die Skalar- und Vektoradditionsfunktionseinheit verarbeiten 64 Bit Integerzahlen. Ganze Zahlen, dargestellt in 64 Bit, liegen bei der CRAY im Bereich

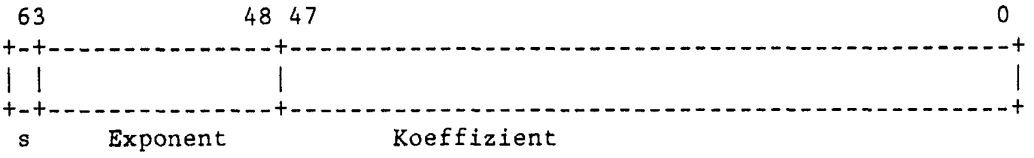
-2^{63} (-9.223.372.036.854.775.807) < = N < = 2^{63-1} (9.223.372.036.854.775.806) (2^{63} ungefähr = 10^{18})

Der alte FORTRAN-Compiler CFT verwendet 24 Bit und 64 Bit Integer, CFT77 verwendet 46 Bit (fast mode) und 64 Bit (slow mode) Integer. Die Auswahl erfolgt über Compiler-Optionen.
Bei CFT werden die Indizes von DO-Loops oder von Feldelementen in den Adressfunktionseinheiten berechnet und dürfen deshalb 2^{23}-1 nicht übersteigen.

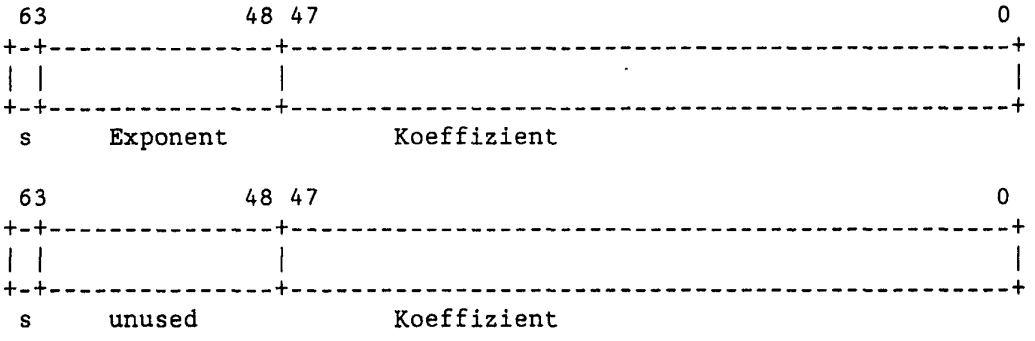
C.2 Gleitkommazahlen (REAL)

Eine Gleitkommazahl wird normalerweise als Dezimalbruch multipliziert mit einem Exponent zur Basis 10 dargestellt. Beispielsweise wird 3580.00 zu $0.3580*10^4$. Eine interne Gleitkommazahl an der CRAY wird aufgespalten in einen binären Koeffizienten und einen Exponenten mit der Basis zwei. Der Koeffizient ist eine 48-Bit-Mantisse. Das Vorzeichen der Mantisse steht in Bit 63. Ein negativer Koeffizient wird nicht komplementiert, sondern der absolute Wert steht als Binärzahl im Koeffizientenfeld und nur das Vorzeichenbit wird gesetzt.

Einfach genaue Gleitkommazahl:



Doppelt genaue Gleitkommazahl:



s = Vorzeichen des Koeffizienten

Die Darstellung von Gleitkommazahlen:

Der Exponent steht als geschobene (biased) Integerzahl in Bit 48 bis 62. In den zur Verfügung stehenden 15 Bits kann (oktal) ein Wert im Bereich

00000 bis 77777

dargestellt werden. Darin liegt der Exponent, der sowohl positiv als auch negativ sein kann. Die Mitte des Bereiches (40000) wird einem Exponenten mit dem Wert null (2^0) gleichgesetzt. Zu jedem Exponenten wird also 40000 addiert, damit diese Darstellung erreicht wird. Der dem Benutzer zur Verfügung stehende Bereich für einen Exponenten liegt zwischen:

20000 bis 57777 (verschobener Exponent)

oder -20000 bis +17777 (echter Exponent)

also -220000 bis 217777 (Zahlbereich)

Werte unter 20000 und über 57777 werden benutzt, Überläufe "underflow" und "overflow" darzustellen.

Die interne Darstellung von Überläufen:

Wenn Bit 62 und 61 gleich sind, so bedeutet das einen Überlauf.

		Bit: 62 61 60
00000		0 0 0
:	underflow	
17777		0 0 1

20000	-20000	0 1 0
:		
40000	0	1 0 0
:		
57777	17777	1 0 1

60000		1 1 0
:	overflow	
77777		1 1 1

Gleitkommazahlen bei CRAY liegen im Bereich von ungefähr

$0{,}458401*10^{-2466}$ bis $0{,}545374*10^{2466}$

Bei einfacher Genauigkeit kann bei der 48-Bit Mantisse mit 14 signifikanten Dezimalstellen gearbeitet werden. Bei doppelter Genauigkeit und 2*48-Bit-Mantisse erhöht sich die Anzahl der signifikanten Stellen auf 29.

Das folgende Bild zeigt ein Beispiel, wie eine REAL-Zahl intern gespeichert ist:

11,0	=	0	4 000 4	5 4	00000000000000
		+	2^4	101 100 000 ... 000	
			16	$2^{-1}+0*2^{-2}+2^{-3}+2^{-4}$ ...	
				0,6875	
11,0	=	+	16	*	0,6875

C.3 Logische Größen

Logische Variablen bei CRAY-FORTRAN (CFT77, CFT) können die Werte TRUE und FALSE annehmen. TRUE wird intern als ein negativer Wert und FALSE als Null oder ein positiver Wert dargestellt. Anders gesagt: Wenn das Vorzeichenbit 1 ist, so ist der Wert TRUE; wenn das Vorzeichenbit 0 ist, so ist der Wert FALSE.

Anhang D. Zeichendarstellung

Zeichen (CHARACTER) werden auf der CRAY in ASCII dargestellt. Für die Darstellung eines Zeichens werden 7 Bit und ein zusätzliches Kontrollbit verwendet; in ein CRAY-Wort passen also acht Zeichen. In der folgenden Tabelle stehen die 128 möglichen ASCII-Zeichen, sowohl druckende Zeichen als auch Steuerzeichen; hier steht die Kurzschreibweise ^ für <CTRL>. Der Darstellung des ASCII-CODE ist oktal, ohne Kontrollbit.

ZEICHEN	ASCII-CODE	BESCHREIBUNG	
<nul>	000	Taste ^@	
<soh>	001	Taste ^A	
<stx>	002	Taste ^B	
<etx>	003	Taste ^C	
<eot>	004	Taste ^D	
<enq>	005	Taste ^E	
<ack>	006	Taste ^F	
<bel>	007	Taste ^G	
<bs>	010	Taste <BS> oder ^H	
<ht>	011	Taste <TAB> oder ^I	
<lf>	012	Taste <LF> oder ^J	
<vt>	013	Taste <VT> oder ^K	
<ff>	014	Taste <FF> oder ^L	
<cr>	015	Taste <RET>	
<so>	016	Taste ^N	
<si>	017	Taste ^O	
<dle>	020	Taste <DLE> oder ^P	
<dc1>	021	Taste ^Q	
<dc2>	022	Taste ^R	
<dc3>	023	Taste ^S	
<dc4>	024	Taste ^T	
<nak>	025	Taste ^U	
<syn>	026	Taste ^V	
<etb>	027	Taste ^W	
<can>	030	Taste ^X	
	031	Taste ^Y	
<sub>	032	Taste ^Z	
<esc>	033	Taste <ESC>	
<fs>	034	Taste ^\	
<gs>	035	Taste ^]	
<rs>	036	Taste ^ ~	
<us>	037	Taste ^/	
"blank"	040	Leerzeichen	(space)
!	041	Ausrufezeichen	(exclamation mark)
"	042	Anführungszeichen	(quotation mark / double quotes)
#	043	Balkenkreuz	(number sign)
$	044	Dollar	(dollar sign)
%	045	Prozent	(percent)
&	046	kaufmännisches "und"	(ampersand)
'	047	Hochkomma / Accent aigu	(apostrophe / single quotes)
(	050	offene Klammer	(left parenthesis)
)	051	geschlossene Klammer	(right parenthesis)
*	052	Stern	(asterisk)
+	053	Pluszeichen	

ZEICHEN	ASCII-CODE	BESCHREIBUNG	
,	054	Komma	(comma)
-	055	Minuszeichen	(minus / hyphen)
.	056	Punkt	(period)
/	057	Schrägstrich	(slash)
0	060	null	
1	061	eins	
2	062	zwei	
3	063	drei	
4	064	vier	
5	065	fünf	
6	066	sechs	
7	067	sieben	
8	070	acht	
9	071	neun	
:	072	Doppelpunkt	(colon)
;	073	Strichpunkt	(semicolon)
<	074	Kleinerzeichen	(less than)
=	075	Gleichheitszeichen	(equal sign)
>	076	Größerzeichen	(greater than)
?	077	Fragezeichen	(question mark)
@	100	"Klammeraffe"	(commercial "at")
A	101	\	
B	102	|	
C	103	|	
D	104	|	
E	105	|	
F	106	|	
G	107	|	
H	110	|	
I	111	|	
J	112	|	
K	113	|	
L	114	|	
M	115	| > Großbuchstaben	
N	116	|	
O	117	|	
P	120	|	
Q	121	|	
R	122	|	
S	123	|	
T	124	|	
U	125	|	
V	126	|	
W	127	|	
X	130	|	
Y	131	|	
Z	132	/	
[	133	eckige Klammer auf	(left bracket)
\	134		(backslash)
]	135	eckige Klammer zu	(right bracket)
^	136	Accent circonflexe	(circumflex, caret symbol)
_	137	Unterstreichung	(underline)
`	140	Accent grave	(back quotes)

ZEICHEN	ASCII-CODE	BESCHREIBUNG	
a	141	\	
b	142	|	
c	143	|	
d	144	|	
e	145	|	
f	146	|	
g	147	|	
h	150	|	
i	151	|	
j	152	|	
k	153	|	
l	154	|	
m	155	| > Kleinbuchstaben	
n	156	|	
o	157	|	
p	160	|	
q	161	|	
r	162	|	
s	163	|	
t	164	|	
u	165	|	
v	166	|	
w	167	|	
x	170	|	
y	171	|	
z	172	/	
{	173	geschweifte Klammer auf	(left brace)
|	174	senkrechter Strich	(vertical line)
}	175	geschweifte Klammer zu	(right brace)
~	176		tilde
DEL	177	delete (Steuerzeichen)	

SEHR GEEHRTER LESER,

wir freuen uns, wenn Sie bei der Verbesserung unseres Handbuches behilflich sind. Für Mitteilungen an uns haben wir, dem amerikanischen Beispiel folgend, die folgenden Blätter als Brief an uns vorbereitet: bitte trennen Sie dazu eines davon heraus und schreiben auf, wenn Sie z.B. -

* Fehler gefunden haben,

* Erweiterungen oder Streichungen vorschlagen,

* einzelne Teile diese Handbuchs allgemein kommentieren möchten.

Bitte falten Sie dann das Blatt an der jeweiligen Strich-Markierung nach hinten, stecken es in einen Fensterumschlag und schicken es an unsere bereits eingetragene Adresse.

Vielen Dank!

Absender:

An das
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
zu H. von Herrn Wolfgang Stech
Heilbronner Straße 10

1000 Berlin 31

Datum:

Betr.: CRAY-Handbuch des ZIB, Kapitel ____, Seite ____, ____. Absatz

Sehr geehrter Herr Stech,

Absender:

An das
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
zu H. von Herrn Wolfgang Stech
Heilbronner Straße 10

1000 Berlin 31

Datum:

Betr.: CRAY-Handbuch des ZIB, Kapitel ____, Seite ____, ____. Absatz

Sehr geehrter Herr Stech,

Absender:

An das
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
zu H. von Herrn Wolfgang Stech
Heilbronner Straße 10

1000 Berlin 31

Datum:

Betr.: CRAY-Handbuch des ZIB, Kapitel ____, Seite ____, ____. Absatz

Sehr geehrter Herr Stech,

Absender:

An das
Konrad-Zuse-Zentrum für
Informationstechnik Berlin
zu H. von Herrn Wolfgang Stech
Heilbronner Straße 10

1000 Berlin 31

Datum:

Betr.: CRAY-Handbuch des ZIB, Kapitel ____, Seite ____, ____. Absatz

Sehr geehrter Herr Stech,

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Technical Reports **Juni 1990**

TR 86-1. H. J. Schuster. *Tätigkeitsbericht (vergriffen)*

TR 87-1. Hubert Busch; Uwe Pöhle; Wolfgang Stech. *CRAY-Handbuch. - Einführung in die Benutzung der CRAY.*
TR 87-2. Herbert Melenk; Winfried Neun. *Portable Standard LISP Implementation for CRAY X–MP Computers. Release of PSL 3.4 for COS.*
TR 87-3. Herbert Melenk; Winfried Neun. *Portable Common LISP Subset Implementation for CRAY X–MP Computers.*
TR 87-4. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / X-MP Systems Running COS Version 3.3*
TR 87-5. Herbert Melenk; Winfried Neun. *REDUCE Users Guide for the CRAY 1 / X-MP Series Running COS. Version 3.3*
TR 87-6. Rainer Buhtz; Jens Langendorf; Olaf Paetsch; Danuta Anna Buhtz. *ZUGRIFF - Eine vereinheitlichte Datenspezifikation für graphische Darstellungen und ihre graphische Aufbereitung.*
TR 87-7. J. Langendorf; O. Paetsch. *GRAZIL (Graphical ZIB Language).*

TR 88-1. Rainer Buhtz; Danuta Anna Buhtz. *TDLG 3.1 - Ein interaktives Programm zur Darstellung dreidimensionaler Modelle auf Rastergraphikgeräten.*
TR 88-2. Herbert Melenk; Winfried Neun. *REDUCE User's Guide for the CRAY 1 / CRAY X-MP Series Running UNICOS. Version 3.3.*
TR 88-3. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / CRAY X-MP Systems Running UNICOS. Version 3.3.*
TR 88-4. Danuta Anna Buhtz; Jens Langendorf; Olaf Paetsch. *GRAZIL-3D. Ein graphisches Anwendungsprogramm zur Darstellung von Kurven- und Funktionsverläufen im räumlichen Koordinatensystem.*
TR 88-5. Gerhard Maierhöfer; Georg Skorobohatyj. *Parallel-TRAPEX. Ein paralleler, adaptiver Algorithmus zur numerischen Integration ; seine Implementierung für SUPRENUM-artige Architekturen mit SUSI.*

TR 89-1. *CRAY-HANDBUCH. Einführung in die Benutzung der CRAY X-MP unter UNICOS.*
TR 89-2. Peter Deuflhard. *Numerik von Anfangswertmethoden für gewöhnliche Differential-*gleichungen.
TR 89-3. Artur Rudolf Walter. *Ein Finite-Element-Verfahren zur numerischen Lösung von Erhaltungsgleichungen.*
TR 89-4. Rainer Roitzsch. *KASKADE User's Manual.*
TR 89-5. Rainer Roitzsch. *KASKADE Programmer's Manual.*
TR 89-6. Herbert Melenk; Winfried Neun. *Implementation of Portable Standard LISP for the SPARC Processor.*
TR 89-7. Folkmar A. Bornemann. *Adaptive multilevel discretization in time and space for parabolic partial differential equations.*
TR 89-8. Gerhard Maierhöfer; Georg Skorobohatyj. *Implementierung des parallelen TRAPEX auf Transputern.*

TR 90-1. Karin Gatermann. *Gruppentheoretische Konstruktion von symmetrischen Kubaturformeln.*
TR 90-2. Gerhard Maierhöfer; Georg Skorobohatyj. *Implementierung von parallelen Versionen der Gleichungslöser EULEX und EULSIM auf Transputern.*
TR 90-3. *CRAY-Handbuch. Einführung in die Benutzung der CRAY X-MP unter UNICOS 5.1*
TR 90-4. Hans-Christian Hege. *Datenabhängigkeitsanalyse und Programmtransformationen auf CRAY-Rechnern mit dem Fortran-Präprozessor* ***fpp.***

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Preprints Juni 1990

SC 86-1. P. Deuflhard; U. Nowak. *Efficient Numerical Simulation and Identification of Large Chemical Reaction Systems.* (vergriffen) In: Ber. Bunsenges. Phys. Chem., vol. 90, 1986, 940-946
SC 86-2. H. Melenk; W. Neun. *Portable Standard LISP for CRAY X-MP Computers.*

SC 87-1. J. Anderson; W. Galway; R. Kessler; H. Melenk; W. Neun. *The Implementation and Optimization of Portable Standard LISP for the CRAY.*
SC 87-2. Randolph E. Bank; Todd F. Dupont; Harry Yserentant. *The Hierarchical Basis Multigrid Method.* (vergriffen) In: Numerische Mathematik, 52, 1988, 427-458.
SC 87-3. Peter Deuflhard. *Uniqueness Theorems for Stiff ODE Initial Value Problems.*
SC 87-4. Rainer Buhtz. *CGM-Concepts and their Realizations.*
SC 87-5. P. Deuflhard. *A Note on Extrapolation Methods for Second Order ODE Systems.*
SC 87-6. Harry Yserentant. *Preconditioning Indefinite Discretization Matrices.*

SC 88-1. Winfried Neun; Herbert Melenk. *Implementation of the LISP-Arbitrary Precision Arithmetic for a Vector Processor.*
SC 88-2. H. Melenk; H. M. Möller; W. Neun. *On Gröbner Bases Computation on a Supercomputer Using REDUCE.* (vergriffen)
SC 88-3. J. C. Alexander; B. Fiedler. *Global Decoupling of Coupled Symmetric Oscillators.*
SC 88-4. Herbert Melenk; Winfried Neun. *Parallel Polynomial Operations in the Buchberger Algorithm.*
SC 88-5. P. Deuflhard; P. Leinen; H. Yserentant. *Concepts of an Adaptive Hierarchical Finite Element Code.*
SC 88-6. P. Deuflhard; M. Wulkow. *Computational Treatment of Polyreaction Kinetics by Orthogonal Polynomials of a Discrete Variable.* (vergriffen) In: IMPACT, 1, 1989, 269-301.
SC 88-7. H. Melenk; H. M. Möller; W. Neun. *Symbolic Solution of Large Stationary Chemical Kinetics Problems.*
SC 88-8. Ronald H. W. Hoppe; Ralf Kornhuber. *Multi-Grid Solution of Two Coupled Stefan Equations Arising in Induction Heating of Large Steel Slabs.*
SC 88-9. Ralf Kornhuber; Rainer Roitzsch. *Adaptive Finite-Element-Methoden für konvektions-dominierte Randwertprobleme bei partiellen Differentialgleichungen.*
SC 88-10. S -N. Chow; B. Deng; B. Fiedler. *Homoclinic Bifurcation at Resonant Eigenvalues.*

SC 89-1. Hongyuan Zha. *A Numerical Algorithm for Computing the Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-2. Hongyuan Zha. *Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-3. Wu Huamo. *On the Possible Accuracy of TVD Schemes.*
SC 89-4. H. Michael Möller. *Multivariate Rational Interpolation: Reconstruction of Rational Functions.*
SC 89-5. Ralf Kornhuber; Rainer Roitzsch. *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers.*
SC 89-6. Wu Huamo; Yang Shuli. *MmB-A New Class of Accurate High Resolution Schemes for Conservation Laws in Two Dimensions.*
SC 89-7. U. Budde; M. Wulkow. *Computation of Molecular Weight Distributions for Free Radical Polymerization Systems.*
SC 89-8. Gerhard Maierhöfer. *Ein paralleler adaptiver Algorithmus für die numerische Integration.*
SC 89-9. Harry Yserentant. *Two Preconditioners Based on the Multi-Level Splitting of Finite Element Spaces.*
SC 89-10. Ronald H. W. Hoppe. *Numerical Solution of Multicomponent Alloy Solidification by Multi-Grid Techniques.*

SC 90-1. M. Wulkow; P. Deuflhard. *Towards an Efficient Computational Treatment of Heterogeneous Polymer Reactions.*
SC 90-2. Peter Deuflhard. *Global Inexact Newton Methods for Very Large Scale Nonlinear Problems.*
SC 90-3. Karin Gatermann. *Symbolic solution of polynomial equation systems with symmetry.*
SC 90-4. Folkmar A. Bornemann. *An Adaptive Multilevel Approach to Parabolic Equations I. General Theory & 1D-Implementation.*
SC 90-5. P. Deuflhard; R. Freund; A. Walter. *Fast Secant Methods for the Iterative Solution of Large Nonsymmetric Linear System.*
SC 90-6. Daoliu Wang. *On Symplectic Difference Schemes for Hamiltonian Systems.*
SC 90-7. P. Deuflhard; U. Nowak; M. Wulkow. *Recent Developments in Chemical Computing.*