Herbert Melenk Winfried Neun

REDUCE User's Guide for CRAY Computers Running UNICOS

(Version 3.3)

Technical Report TR 88-2 (February 1988)

Konrad-Zuse-Zentrum für Informationstechnik:

Heilbronner Straße 10; D-1000 Berlin 31

Herbert Melenk Winfried Neun

REDUCE User's Guide for CRAY Computers Running UNICOS

(Version 3.3)

Abstract

This document describes operating procedures for running REDUCE specific to the CRAY-1, CRAY X-MP, CRAY Y-MP and CRAY-2 computers running the Operating System UNICOS. The document was derived from the corresponding document for Vax/UNIX prepared by Dr. A. C. Hearn and L. R. Seward, The Rand Corporation, Santa Monica, (CP85).

Contents

1. Preliminary

This document describes operating procedures for running REDUCE specific to the CRAY-1, CRAY X-MP, CRAY Y-MP and CRAY-2 computers with the UNICOS operating system. It supplements the REDUCE User's Manual, describing features, extensions and limitations specific to this implementation of REDUCE.

The character ! (exclamation mark) is used in this document to represent the REDUCE syntax escape character. By using REDUCE on a CRAY computer via a station computer this character (and others too) may have a different representation due to local code conversions. Please look at the special character table in the REDUCE User's Manual.

The modules that form the REDUCE system are stored in a number of files. The main entry to REDUCE is the executable file named "reduce". Maybe you must adapt your environment's access path. At runtime the system needs access to REDUCE and LISP libraries. This access is achieved in an automatic and for the user invisible manner. To start REDUCE, simply use the command

```
reduce
```

in your batch job or interactive session, after which REDUCE will respond with a banner line. The system then expects its input via the standard input file and writes to standard output file.

Examples for batch REDUCE jobs:

```
reduce ≪EOF                  reduce < reducein > reduceout
p := (x + y) * *10;          (where reducein contains 2 lines:
bye;                         p := (x + y) * *10;
EOF                          bye;)
```

Note: the lower characters are converted to capital characters on input automatically by default (raise is on).

2. Reduce Documentation

For proper usage of REDUCE, the document

A.C. Hearn:
REDUCE User's Manual

should be consulted. As an addendum to the User's Manual, a number of modules first introduced by REDUCE 3.3 are described separately. If features of the underlying LISP system are to be used, the documents

Utah Symbolic Computation Group:
The Portable Standard LISP Users Manual

and

H. Melenk, W. Neun:
Portable Standard LISP Implementation for CRAY X-MP computers

may be needed. All these documents are distributed by Konrad-Zuse-Zentrum für Informationstechnik Berlin. There is no on-line documentation available.

3. An Introduction to Reduce

New users of REDUCE are advised to process the seven REDUCE Lessons, which should be used via a REDUCE implementation on a timesharing computer or a workstation (e.g. a VAX or a SUN). They are not available on-line with CRAY computers.

4. Resource Requirements

The minimum field length of a job running REDUCE is approximately 500 k words. In this field length the basic REDUCE features can be loaded; it includes a heap of total length 50000 words with about 40000 words available as free working memory. If additional modules of REDUCE are loaded at runtime or if due to the problem size a larger heap is needed, more memory can be allocated and the memory parameter has to be modified (in case of a batch job).

The areas for heap (data memory), bps (area for compiled code), stack and binding stack (BNDSTK) can be enlarged up to the limit defined by the memory parameter.

The enlargement of binary program space while loading additional modules or compiling user programs is done automatically (the action is protocoled via standard output).

At the end of a REDUCE run you will get a statistic from the underlying PSL which looks like:

```
         PSL 3.4        22.08.88     12:50:51
     Total cpu time:   9710 ms      heapsize:  200000
Garbage collections:      8         bpssize:   150000
       Time for gc:    1224 ms     stacksize:    5000
 Ratio: gc / total:      12  %    bndstksize:     500
```

The block on the left side summarizes the cpu time needed to run the REDUCE application, the time for garbage collection, which is included in this total cpu time, the number of garbage collections and the percentage of cpu time spend for garbage collection are printed below. A greater heapsize will normally result in a better cpu/gc time percentage. The block on the left contains the memory requirements for the LISP memory areas that can be adjusted at runtime.

If a larger heap is needed because the number of garbage collections is too high, the LISP function SET-HEAP-SIZE has to be called. This function expects the desired TOTAL heap length as parameter. It is called from REDUCE e.g. by

```
LISP SET!-HEAP!-SIZE 100000;
```

(Note the exclamation marks in front of special characters inside the name.) The heap size is adjusted in case of a lack of data area too, but this will happen if the heap is almost full, after many garbage collections. So we recommend to adjust heapsize when big data areas are needed.

The actual total heap size can be seen as value of the LISP variable heapsize via the REDUCE command: LISP heapsize;

The PSL parameters set by REDUCE normally condemn the garbage collector to do his work in total silence. This is necessary because asynchronous messages can destroy a handsome output image. In order to learn the amount of occupied heap space or in order to see at which points of execution the garbage collector is involved, simply turn on the switch: ON GC;

With GC on the garbage collector will be verbose. If you only want to know the number of garbage collections during your calculation you ask for the value of the LISP variable

LISP GCKNT!*; % number of garbage collections since start;

It would be a good strategy to inspect this variable after a large computation has been done the first time and to adjust the heap size so that this value remains in a reasonable range. For memory estimates: the underlying PSL uses two words to store one list element (=two pointers).

In case of stack overflow the stack size can be modified in a corresponding manner by calling the function SET!-STACK!-SIZE. The actual size of the stack is the value of the LISP variable StackSize. A similar function SET!-BNDSTK!-SIZE is supplied for BNDSTK.

There is no way to resume a calculation that lead to an overflow of stack or bndstk after enlarging the memory areas.

5. File Handling

REDUCE file handling is adapted to the UNICOS environment, such that the user has all possibilities to 'navigate' in the file system. For simple applications the following notes may be sufficient:

- Usage of files in the current working directory:

  Most easy access is to files whose names that consist of lowercase letters and digits only. Other filenames containing capital letters or special characters have to be included in quotation marks, e.g.: **in myfile**; or **IN MyFile**; will read **myfile** in current directory but in **"HUGO"**; or in **"input.dat"**; will **read** files HUGO or input.dat respectively. (Note that **IN HUGO**; will read file **hugo**.)

- Access to files in other directories or on a frontend system
  There are different ways to access files that are not in the current directory:

  - Absolute addressing via full path. e.g. in "/tmp/input.red";
  - Relative addressing using environment variables; e.g.
    in " ~hugo/input.red"; or **in "$HOME/data"**;
  - Changing pwd via the command LISP UNICOS!-CHDIR "~hugo"; before file is opened.

- Staging a file from a frontend machine before opening it, e.g.:

  system "rcp sn4711:kuno/file.red newinput.red";

  (Note: system is an alias for LISP SHELLCMD, which invokes *csh* to handle the command given as parameter string.)

- Usage of Libraries
  Libraries contain binary files, which can be loaded into REDUCE at run-time to provide features which are not included in the REDUCE main program, e.g. the PSL compiler. The REDUCE load module library is opened at startup time automatically. Additional libraries can be accessed using the commands:

  ```
  LISP set!-libpath "dirstring";
  Libopen lib;
  ```

  Where dirstring is the name of the directory containing lib. Default for dirstring is " " i.e. the current working directory is assumed to contain the library.

6. Internal Parameters

6.1 Object Sizes

The maximum string and identifier lengths are limited only by the underlying PSL base. The current implementation allows several hundred characters in both identifiers and strings. However, we recommend that such names be limited to 24 characters or less for compatibility with other versions of REDUCE.

All fixed precision floating point numbers are printed in FORTRAN's "G23.8E4" format by default. This format may be changed as described in the PSL documentation.

Arbitrary precision integer and real arithmetic is supported.

6.2 Special Characters and Interrupts

Lower case input is permitted, but converted to upper case unless the switch RAISE is off.

^ may be used as an alternative to ** in expressions. This character is represented by the character "not" with some front-end computers.

The end-of-file is supplied by the *control-D character (hex 04)* in interactive mode.

If an interrupt occurs (e.g. by entering *control-C (hex 03))*, the current calculation is cancelled and a little diagnosis of the problem is printed by the underlying PSL system. REDUCE immediately prompts for the next command. There is no means to continue the interrupted calculation. The LISP function USER!-ERROR!-FUNCTION is called in recovery procedure. It may be redefined for own diagnostics. The signal number is the only parameter, e.g. 2 for terminal interrupt.

} is used to terminate strings in the REDUCE interactive editor, because ESC as in other implementations will not be handled via all stations.

6.3 Miscellaneous

Several UNICOS funtion can be invoked from REDUCE to alter the REDUCE environment in contrary to a shell invocation via SYSTEM command, which does not alter the REDUCE environment. These functions are:

Unicos!-pwd(), Unicos!-getenv "var", Unicos!-putenv "var=value", and Unicos!-chdir "directory".

There is no link currently to an external editor. The internal ordering on alphabetic characters is from A through Z followed by a through z.

Times (as reported by ON TIME or SHOWTIME) are given in milliseconds, and measure execution time including garbage collection time. They do not include operating system overhead.

To exit REDUCE use "bye;" .

Per default there is echoing of input in batch jobs only. Echoing is controlled by the switch

```
on echo;
```

With echo turned on, the input is echoed to the output and a mixed protocol of incoming commands and outgoing results of their execution is produced.

In batch jobs the switch int is set to off by default. This switch indicates the batch/interactive state to the REDUCE system.

With int turned off REDUCE will generate no interactive requests and will stop algebraic execution when the first error occurs. To avoid this, you can use ON

ERRCONT; even in batch jobs.

7. Customizing the REDUCE Environment

Implementation deferred. Datasets needed at startup time have to be processed explicitly.

8. Implementation Dependent Messages

A number of messages from the underlying PSL system and the CRAY machine interface may be seen from time to time. These include error messages and informative messages.

- Signal 8 invoked: Floating point exception
  Probably means a division by zero has been attempted.
- BPS will be automatically enlarged ... (Information)
  BPS size is increased because a load module was needed by your application, e.g. the PSL compiler.
- Heap space will be automatically enlarged... (Warning)
  Your problem is too large in its present form for the available workspace; either change your problem formulation or enlarge your heap (see above)
- Non-numeric argument in arithmetic
  This means that a LISP arithmetic routine has been called with an invalid argument.
  Hint: If you do not know what happened use
  TR CONTINUABLEERROR; and run again.
- Signal 11 invoked: Operand range error
  This indicates an illegal memory reference. It can arise from applying the LISP function CAR to an atom in compiled code.
- Signal 26 invoked: cpu time limit exceeded
  The time parameter was too small for your batch job.
- Stack overflow
  The PSL stack has overflown. If your application needs a larger stack, please enlarge it via a LISP Set!-Stack!-size <Size>; call.
- Binding stack overflow
  The PSL stack for special variables has overflown. If your application needs a larger BNDSTK, please use a LISP Set!-Bndstk!-size <size>; call.

A hint: if you want a backtrace in case of an "signal *xx*", type

```
SYMBOLIC PROCEDURE USER!-ERROR!-FUNCTION(x);
              INTERPBACKTRACE();
```

and run again.

9. Further Help

For further help with the CRAY implementation of REDUCE, please contact:

Konrad-Zuse-Zentrum für Informationstechnik Berlin
email: zb6260 @ db0zib21 · bitnet