LA-UR -86-3771

LA-UR--86-3771

DE87 002921

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: A UNIX INTERFACE TO SUPERCOMPUTERS

AUTHOR(S): Oliver A. McBryan*

SUBMITTED TO: Proceedings of ARO Meeting on
Workstations and Supercomputing
Newark, Delaware
May 1985

*C-3 Collaborator from Courant Institute of Mathematical Sciences, New York University, New York, NY 10012.

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

A UNIX Interface to Supercomputers[1]

Oliver A. McBryan[2, 3]

Los Alamos National Laboratory,[4]
Los Alamos, NM 87545.

ABSTRACT

We describe a convenient interface between UNIX-based work-stations or minicomputers, and supercomputers such as the CRAY series machines. Using this interface, the user can issue commands entirely on the UNIX system, with remote compilation, loading and execution performed on the supercomputer. The interface is not a remote login interface. Rather the domain of various UNIX utilities such as compilers, archivers and loaders are extended to include the CRAY. The user need know essentially nothing about the CRAY operating system, commands or filename restrictions. Standard UNIX utilities will perform CRAY operations transparently. UNIX command names and arguments are mapped to corresponding CRAY equivalents, suitable options are selected as needed, UNIX directory tree filenames are coerced to allowable CRAY names and all source and output files are automatically transferred between the machines.

The primary purpose of the software is to allow the programmer to benefit from the interactive features of UNIX systems including screen editors, software maintenance utilities such as *make* and *SCCS* and in general to avail of the large set of UNIX text manipulation features. The interface was designed particularly to support development of very large multi-file programs, possibly consisting of hundreds of files and hundreds of thousands of lines of code. All CRAY source is kept on the work-station. We have found that using the software, the complete program development phase for a large CRAY application may be performed *entirely* on a work-station.

---

1. Presented to the ARO meeting on Work-stations and Supercomputing, Newark, Deleware, May 1985.
2. Supported in part by DOE contract DE-AC02-76ER03077
3. Supported in part by NSF grant DMS-83-12229
4. Permanent address: Courant Institute of Mathematical Sciences, New York University, New York, N.Y. 10012.

A UNIX Interface to Supercomputers[1]

Oliver A. McBryan[2, 3]

Los Alamos National Laboratory,[4]
Los Alamos, NM 87545.

1. Introduction

We have developed UNIX-based software which provides a UNIX workstation or minicomputer user with transparent access to a CRAY or other supercomputer. Effectively, we extend the domain of certain UNIX utilities to the supercomputer. It is assumed that there is a direct connection between the workstation and the supercomputer. The connection should be high-speed for reasonable efficiency since remote source file transfers are involved. Facilities supported include remote compilation, linking and execution, along with data retrieval.

The goal of the software described here is to allow the user to develop new programs entirely in a UNIX work-station environment. Once fully developed, the programs are generally run directly on the CRAY, although remote execution is also available.

The primary observation that led to development of this software was that 95% of our CRAY program development time was involved with routine editing, debugging and compilation activities. Most of these activities are best performed on a workstation. For example, powerful software maintenance utilities such as *make* and *SCCS* are not available on the CRAY, and can greatly speed the program development effort, especially for large multi-file programs. As a direct application, we can now use *make*

1. Presented to the ARO meeting on Work-stations and Supercomputing, Newark, Deleware, May 1985.
2. Supported in part by DOE contract DE-AC02-76ER03077. 3. Supported in part by NSF grant DMS-83-12229
4. Permanent address: Courant Institute of Mathematical Sciences, New York University, New York, N.Y. 10012.

to maintain large programs on the CRAY.

There are many additional advantages to distributed program development. The workstation user can avail of full interactivity, to an extent not available on supercomputers. In particular one typically finds faster response on work-stations to interrupt-driven facilities such as editors. Various powerful software tools, including screen editors, high-level languages and transformational utilities are available on work-stations. Furthermore by off-loading interactive activity, the supercomputer is freed for batch processing, which is where it performs best. In many cases the work-station may also be used effectively for graphical post-processing of data returned from the supercomputer.

Section 2 provides an overview of the facilities supported by the interface software. Section 3 describes in detail the filename coercion facilities that are used to map the UNIX filename space into the more restrictive filename space of the supercomputer. Sections 4, 5 and 6 describe the mapping of basic compilation utilities to their CRAY equivalents. Finally section 7 discusses extension of the domain of the *make* utility to include the supercomputer.

2. Scope and Facilities

We will discuss the UNIX interface in terms of CRAY computers running the CTSS operating system, although we have implemented a similar interface to the COS operating system. The general mechanism is clearly extendible to other supercomputers. The network connection between the supercomputer and the work-station is also a factor - we discuss here the use of facilities of the Los Alamos Integrated Computing Network. However the network facilities required are so simple that a similar system could likely be built on top of any reasonable network. In fact, the existence of a file transfer protocol would suffice, using software we have described in a related paper.[1]

The fundamental approach we have taken is to implement only the most frequently used CRAY utilities as UNIX utilities. Shell command files are created that implement the desired CRAY commands as UNIX utilities, taking standard UNIX arguments, options and filenames. These command files map the UNIX commands into their corresponding CRAY equivalents, supply the appropriate options to the CRAY command, arrange that any file arguments are transferred to the CRAY, and when everything is in place, execute the correct CRAY command on the remote machine. After command execution, any output or error messages are returned to the work-station and are directed to the user's standard output.

An important aspect of the steps described above is the transfer of files. File formats usually need to be modified on each side before and after transfer. More significantly, the supercomputer and work-station will generally use different file name spaces. To provide maximum generality we provide for filename coercion between the systems. The filename coercion facilities are the same among the various basic utilities. Furthermore uniform conventions are adopted for file location. All source files (including assembler) are kept on the UNIX machine while all object, library and executables are kept on the CRAY. Readable output files such as assembler source or compiler listings are returned with an appropriate name to the UNIX machine. In cases where files are returned, the inverse of the filename coercion function is applied providing a reasonable UNIX expansion. We discuss all of these issues in more detail below.

In our case the most important target utilities are the CRAY Fortran compiler *CFT*, the CRAY C compiler *CC*, the CRAY assembler *CAL*, the CRAY loader *LDR*, and the CRAY Librarian *BUILD*. The corresponding UNIX utilities are the *f77*, *cc*, *ar* and *ld* programs. Thus we discuss these cases in most detail. Higher level UNIX utilities such as *make* generally issue commands to low level utilities such as those described above. By developing UNIX compatible utilities that call the corresponding

CRAY utilities we therefore effectively extend the domain of *make* to the CRAY.

While the functionality of basic non-interactive utilities tend to map rather well across systems, it frequently happens that a CRAY utility may require an option that has no corresponding UNIX equivalent (see below for examples). In these cases an extra UNIX option is added to the standard UNIX utility argument list. The result is that even a naive user can compile and link CRAY programs without ever logging into the CRAY or reading any CRAY manual, using the same commands or makefiles he would use on a UNIX machine. Occasionaly an extra option or two may be required in order to support some special CRAY feature, but these do not appear in normal usage.

3. Filename Coercion

CRAY CTSS filenames may contain at most 8 characters, whereas UNIX filenames may have essentially arbitrary length, including a directory part. Similar restrictions are found on many other supercomputer operating systems. Each UNIX filename to be compiled should be alphanumeric apart from a directory prefix and a suffix consisting of *.c*, *.f*, *.s*, *.o* or *.a*. We refer to the filename with the directory prefix and the suffix removed as the file *base name*. The directory prefix will be stripped, but remembered, before sending files to the CRAY. Similarly characters beyond 7 in the base name are stripped, the period is deleted from the suffix but the remaining suffix character is maintained. Consequently the CRAY file name always ends in the same suffix character as the UNIX filename. As examples, */usr/me/shortc* maps into the CRAY name *shortc*, while */usr/me/longname.c* would become *longnamc*. One exception to this rule is that library archive base names are truncated to 8 characters and no suffix is added. This is because typically certain system-supplied CRAY libraries will be required. and will normally not have names ending in *a*. Thus the

UNIX archive file */usr/me/mygoodlib.a* would be represented on the CRAY as the library file *mygoodli*.

In cases such as listing, preprocessor output or assembler files which are generated on the CRAY, files are returned with an inverse coercion rule applied. Such files will be placed in the current working directory with the full basename and an appropriate suffix of *.l*, *.e* or *.s* respectively.

In addition to filename coercion, certain other transformations may be required in exchanging files between the CRAY and work-station. For example, on the Los Alamos network a uniform *standard text* file format, (*stext*), is supported to provide portability between machines. However on each individual machine it is necessary to convert files from *stext* form to the *native text* form for that machine, *ntext*, before processing by editors, compilers or other utilities on the machine. System programs *stext* and *ntext* are supplied to convert a native text file on any machine to standard text format and to convert a standard text file to native text format, respectively. Our software automatically performs these format transformations when exchanging text files between machines of differing architectures. All text files will always be converted into native text form on the machine they reside on.

There is a difference in the treatment of source and object filenames which are provided as arguments to supported utilities. Source, including assembler, filename arguments to utilities (suffixes *.c*, *.f* or *.s*) cause the corresponding UNIX files to be sent to the CRAY with filename coercion as above, as well as appropriate text format transformation between native text modes. Object or library filename arguments (suffixes *.o* or *.a*) are interpreted differently. Each *.o* or *.a* file argument is interpreted as denoting a *previously compiled* file or a previously built library *on the CRAY*. The corresponding CRAY object or library filename is obtained using the rules described above. The rationale here is that there seems little point in moving such object files back to the workstation. They are regarded as conceptually residing on the work-

station, however, in that utilities behave in the same way they would if the files had been stored there. In fact the compile utilities *ccc*, *cft* and *car* described below create dummy object or archive files on the work-station corresponding to each file compiled. These dummy files carry the names one would expect on a UNIX system, i.e. filename coercion is not applied, and their main purpose is to provide a map of the current compilation state on the CRAY, including information about the exact compilation time of each CRAY file.

4. The CCC command

The UNIX version of the CRAY C compile command is called *ccc* to distinguish it from the standard UNIX C compiler *cc*. However *ccc* takes the same standard arguments that the *cc* command takes, along with some CRAY specific ones. The calling sequence is:

ccc [-c] [-o name] [-E] [-O] [-Dstring] .. [-Ustring] .. [-Istring] ..
[-ic] [-l] [-V] [-p priority] [-r] file1 file2 ..

The *–c* option specifies compile only, without loading.

The *–o* option assigns the following name to the compiled program.

The *–E* option runs the C preprocessor on each C file leaving the output in the current directory with suffix *.e*.

The *–O* option is ignored.

The *–Idir* option specifies a search directory for include files.

The *–Dstring* and *–Ustring* options implement preprocessor defines and undefines as in the UNIX C compiler.

The *-pc* option specifies that the C pre-processor is to be executed on the CRAY. The default is to execute the pre-processor on the work-station.

The *-l* option places full listing files in the current directory with suffix *.l*.

The *-V* option specifies that all C sources files are to be compiled specially with the *varargs* mechanism.

The *-p* option specifies that CRAY compilation or loading is to be performed at the specified priority level.

The *-r* option causes the compiled program to be run on the CRAY and the output transferred to the UNIX standard output.

The last five options, *-ic*, *-l*, *-V*, *-p* and *-r* are not standard UNIX facilities. They provide access to desirable CRAY C features. In particular the CRAY C compiler requires a special argument *-V* if a source file containing a subroutine with a variable number of arguments is to be compiled. It is also useful to see the CRAY compiler listing - if the *-l* option is supplied then for each compiled source file, a corresponding CRAY listing file will be returned to the UNIX machine with the same filename, but suffix *.l*.

Filenames ending in *.c*, *.f*, *.s*, *.o* or *.a* are assumed to be C source, Fortran source, CAL assembler, previously compiled CRAY object files or CRAY library archives of object files respectively. Each source file is converted to standard text format, moved to the remote machine, converted to native text and compiled. Thus */usr/me/longname.c* is moved to *longnamc* on the CRAY and is compiled to produce *longnamo*. If a listing file is requested it is returned as *longname.l* to UNIX. After each file is compiled, a corresponding dummy file with suffix *.o* is created in the current directory to record the compilation status and time of compilation on the CRAY. In the above example a dummy file *longname.o* would be created on the work-station.

One issue not discussed so far is the use of the C pre-processor. Since the pre-processor is simply a text transformer, it may obviously be executed either on the work-station or on the CRAY. The *-pc* option is provided to allow the user to choose either possibility. The choice made affects primarily the outcome of the C *include* facility. Depending on which route is taken, file inclusion will be performed either on the CRAY or on the work-station. It is more consistent with our general goals if file inclusion is performed on the work-station - include files are after all text files. However if this is done, certain precautions are required. For example, there are system include files such as *<stdio.h>* which are used by many programs, but are very system-dependent. It would be incorrect to include a work-station version of such a file in source code targeted for the CRAY. Consequently a seperate directory of CRAY system include files must be maintained on the work-station, and searched by the pre-processor before it searches the standard system directory. This is easily accomplished in practice using the *-I* include directory option.

One further pre-processing step is automatically inserted by the *ccc* command, and is provided for two reasons. A disadvantage of the distributed compilation discussed here is that there is delay involved while waiting for files to be transferred to the CRAY. It is therefore very desirable to minimize the length of source files. Secondly, the CRAY C compiler has difficulty with long lines in source files. We handle both of these issues by subjecting each source file to a filter called *shorten* before sending it. This step is performed after pre-processing, if that was requested on the work-station. The *shorten* filter replaces consecutive white-space characters found outside of quotes by a single space, and then folds every line after 79 characters, taking care not to split strings.

If loading is not suppressed by the *-c* option, then both the newly compiled files and the previously compiled files, represented by any *.o* file arguments, are loaded together along with requested libraries. The resulting executable image is called *name*

if the *–o name* option was used, or *a.out* otherwise. Any unrecognized command line arguments are assumed to be options for the CRAY loader. This allows various special facilities to be accessed, for example a dynamic array may be specified in this way to facilitate programs that perform internal storage allocation.

If the *–r* option was specified on the compile line, the compiled program is run and its output returned to the standard output of the UNIX work-station.

5. The CFT Command

The *CFT* command allows Fortran source files to be compiled on a CRAY. The usage is similar to that for the *ccc* command, but with fewer options supported:

cft [-c] [-o name] [-l] [-r] file1.f file2.f filer.o

The *–c* option specifies compilation only.

The *–o* option assigns the following name to the program.

The *–l* option places full listing files in filei.l.

The *–r* option causes the program to be run on the CRAY after loading.

File name coercion, and text file format transformations, are performed in the same way as for the *CCC* command. Thus each Fortran source file (suffix *.f*) is converted to standard text, moved to the remote machine, converted to native text and compiled. The filename file.f will produce a CRAY source file called filef, and a binary file called fileo. The listing file, if requested, is returned to the current directory as file.l. After each file is compiled, a corresponding dummy file with suffix *.o* is created in the current directory on the work-station to record the compilation status and time of compilation on the CRAY. Each object file argument (suffix *.o*) to *cft* is interpreted as denoting a previously compiled object file on the CRAY.

If loading is not suppressed by the *−c* option, then both the newly compiled files and previously compiled files are loaded together and the executable image is named file1x, (derived from the first filename argument), or 'name' if the *−o* option was used. Any remaining command-line arguments are passed to the CRAY loader as arguments.

6. The *CAR* Command

The *CAR* command accesses the CRAY Librarian, *BUILD*, using the standard argument syntax of the UNIX *ar* archive program.

car option libname file1.o file2.o ..

Here *option* is one of *c*, *r*, *t* or *x* denoting respectively *create* a new archive, *replace* files in a library, *list* the table of contents of a library or *extract* all files from a library. Filename coercion follows the rules given earlier in section 3. Thus both the archive name *libname* and the object filenames *filei.o* are subjected to directory and name truncation. All of the resulting object files are assumed to exist on the CRAY and the resulting archive file is also left on the CRAY. After the archive file is created or updated on the CRAY a corresponding dummy file with suffix *.a* is created in the current directory on the work-station to record the archive status and time of archiving on the CRAY.

7. Using MAKE on the CRAY

The real payoff for the development of the facilities described in the previous sections comes when they are coupled with the UNIX *make* program. *Make* is a utility used to maintain software projects. It deals with the mechanics of assembling large

programs from many inter-related source files. The user specifies how to build the program by supplying an appropriate set of commands in a *makefile*. *Make* reads the *makefile*, checks to see what commands remain to be executed to build the required object, and performs these. An important point is that *make* attempts to perform the minimal amount of work necessary to build a program. This is accomplished by using a set of built in dependency rules. For example, *make* realizes that an object file *file.o* is obtained from a source file *file.c* or *file.f*. If *make* is required to create an object file *file.o* as part of building a program, it first checks to see if a corresponding source file *file.c* or *file.f* is available. If so, it compares the date of last modification of the source code file and the last version of the object file (if there is one). If the source code was modified *since* the object file, or if no object file is present, then *make* automatically calls the appropriate compiler command on the source file; otherwise no action is performed by *make* for that object file. For further information about *make* see the UNIX operating system user manuals.[2] For the discussion here we note one other feature of *make*: the built in rules for creating object files from dependencies are easily modified. For example, the rule for creating *file.o* from *file.c* looks like:

$(CC) $(CFLAGS) file.c

Here *CC* and *CCFLAGS* are make macro variables that define the compiler command and compiler options to be used in compiling file.c, while *$(·)* denotes macro variable evaluation. Both are given default values by *make*: *CC = cc* and *CFLAGS = -c* respectively. Similarly, the default rules for compiling Fortran programs involve macro variables *F77* and *FFLAGS*, which have default values of *f77* and *-c*, while the rules for creating a library use a default variable *AR* with default value *ar*.

Suppose that we have a large multi-file, and possibly multi-directory, program maintained for compilation by *make*. Normally the program will be developed and debugged on the work-station. A version for the supercomputer may be maintained from the same source files by using conditional compilation. In this context the C

pre-processor makes an excellent pre-processor for both Fortran and C. We have used it to maintain portable source code for more than 10 machines within a single software version.

Once the work-station version of the program is ready, a supercomputer compilation may be made from the same directory and makefile by simply redefining the *make* variables *CC*, *F77* and *AR* to the values *ccc*, *cft* and *car* respectively. If necessary, the variables *CFLAGS* and *FFLAGS* may be set to record any special options required such as listing files or to request the *-V* option for variable-number-of-argument routines in C. Conditional compilation may also be effected at this point by adding the appropriate conditional compilation flags to *CFLAGS*. *Make* then takes over, issuing all needed instructions for the complete compilation and linking of the program on the CRAY, including handling all of the issues of file transfer and name coercion. While object and library files are not kept on the work-station, the dummy versions of such files are created by the compilers as discussed previously and record faithfully the correct last time of compilation of the corresponding files on the CRAY.

One major inconvenience is that under CTSS and other supercomputer operating systems frequently a user's files are deleted within 24 hours of logging out. As a result it is generally necessary to save all files on the CRAY to a mass storage system before logging off. This is in fact easily automated using a *save* entry in the work-station makefile, which simply looks at the current list of *.o* and *.a* files on the work-station, and sends commands to the CRAY to save those files as well as the corresponding source files. At the next work-station session a *restore* entry in the makefile may be activated to return the CRAY files from storage to the CRAY. However none of these steps involves any actions to the work-station files. The times recorded will still be those for the original CRAY compilation, avoiding the necessity for any unnecessary recompilation.

References

1. O. McBryan, "Using Supercomputers as Attached Processors," Los Alamos National Laboratory Preprint, Sept 1986.

2. *The UNIX Users's Manual Reference Guide*, USENIX Association. 1984.