ARMY RESEARCH LABORATORY

Introductory User's Guide to the ARL Supercomputer Facility at APG

Claude J. Lapointe
Richard Angelini
Lee Ann Brainard
Denice Brown
John Cole
Monte Coleman
Phillip Dykstra
Carol A. Ellis
Gary Kuehl
Deborah L. Thompson
U.S. ARMY RESEARCH LABORATORY

Kathy A. Burke
Jerry A. Clarke
COMPUTER SCIENCES CORPORATION

ARL-TR-150 June 1993

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Final

4. TITLE AND SUBTITLE

Introductory User's Guide to the ARL Supercomputer Facility at APG

5. FUNDING NUMBERS

PR: 4B592-3B2-H2

6. AUTHOR(S)

Claude J. Lapointe, Richard Angelini, Lee Ann Brainard, Denice Brown, John Cole, Monte Coleman, Phillip Dykstra, Carol A. Ellis, Gary Kuehl, Deborah L. Thompson, Kathy A. Burke,* and Jerry A. Clarke*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-CI-AC
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-OP-CI-B (Tech Lib)
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

ARL-TR-150

11. SUPPLEMENTARY NOTES

*Kathy A. Burke and Jerry A. Clarke are employed by Computer Sciences Corporation, 3160 Fairview Park Dr., Falls Church, VA 22042.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of this report is to provide an introductory user's guide to many of the computers, operating systems, utilities, and software packages available at the U.S. Army Research Laboratory Supercomputer Facility located at Aberdeen Proving Ground, Maryland, together with the facility's normal operating procedures.

14. SUBJECT TERMS

ARL Supercomputer Facility, computer user's guide, computer programming, Fortran

15. NUMBER OF PAGES

166

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Intentionally Left Blank

TABLE OF CONTENTS

LIST OF FIGURES

Intentionally left blank

1. Introduction

1.1 Historical Overview

The Army Research Laboratory (ARL) traces its computational history through the Ballistic Research Laboratory (BRL), one of the organizations from which it was formed. The BRL was the home of the world's first electronic digital computer, the ENIAC (**E**lectronic **N**umerical **I**ntegrator **a**nd **C**alculator). The ENIAC, developed from 1943 through 1946 and installed at the BRL in January 1947, was built under contract to assist Army scientists in the calculations of ballistic trajectories, a task that the Army still performs today. Developed at the Moore School of Electrical Engineering of the University of Pennsylvania, the ENIAC contained 19,000 vacuum tubes and 1500 relays, and consumed 200 kilowatts of power. Located in several rooms, the ENIAC was manually programmed by turning 3600 dials and setting numerous switches and plug-in cables. Several weeks of setting up and verification were required for the execution of a major program, a "major program" being the equivalent of 1000 1-address instructions executed on a modern computer.

As a follow-on contract to the ENIAC, the University of Pennsylvania was awarded $105,600 for research and development of a computer that incorporated a "stored program" design. This contract gave birth to the EDVAC (Electronic Discrete Variable Automatic Computer) and to John von Neumann's influence on computers (Stern 1981). The EDVAC, the world's first stored program computer, contained 1024 48-bit words of core memory and provided paper tape input/output and, later, punch card input/output. During EDVAC's life span, the BRL designed and added floating point hardware.

Subsequent to the EDVAC, the ORDVAC (**O**rdnance **V**ariable **A**utomatic **C**omputer) was built in 1952 by the University of Illinois. This computer was operational for 12 years at the BRL and incorporated 4096 40-bit words of core memory, a 10,000-word drum, a magnetic tape drive, and punched tape and card input/output.

The ENIAC through ORDVAC computers provided the cornerstones for the BRL-designed and BRL-built BRLESC (BRL Electronic Scientific Computer) I & II. These "high performance" machines were replaced by a commercial mainframe, the last Cyber 7600, which served the Army from the late 1970s to the middle 1980s.

In the 1980s, the BRL embraced UNIX as an operating system and began to provide a distributed minicomputer environment for BRL scientists. This led to the development of the BRLnet, a campus network with numerous, well populated local area networks (LAN). Currently, these LANs are populated with minicomputers, super-minicomputers, and network printing devices. Additionally, the BRL scientific staff developed a large suite of UNIX utilities and tools to provide a robust, user-friendly, and homogeneous computing environment. Many of the tools and computing techniques developed at the BRL continue to serve as models for other computer centers worldwide.

In 1985, the BRL's director, Dr. Robert J. Eichelberger, recognized that the laboratory was ready for the next generation computer. It was Dr. Eichelberger's intent that the BRL have the best computational tools available. Within 2 years, the BRL had procured two Cray supercomputers. In December 1986, the BRL took delivery of a Cray X–MP/48 with a 128-million-word solid-state disk and in July of 1987 took delivery of a 256-million-word Cray–2. The addition of these machines to the existing array of minicomputers, super-minicomputers, and high-resolution work stations established the laboratory as a high performance computing center.

1.2 Computer Systems and Facilities

The Army Research Laboratory SuperComputer Facility (ARLSCF) has two supercomputers, a Cray X–MP/48 and a Cray–2. Dispersed throughout the ARL technical directorates located at Aberdeen Proving Ground is a collection of minicomputers and super-minicomputers, including VAX 11/780s, VAX 6320s, Alliant FX8s, and CONVEX C1s. Additionally, a wide variety of workstations from vendors such as Sun Microsystems, Silicon Graphics, Apollo, and Stellar complement the general purpose processors.

Local access to this equipment is provided via the BRLnet or MILNET, dialup terminals, and local terminals connected to various Gandalf PACX 1000s. Nationwide access is provided through a number of networks.

The Cray–2 has four central processing units (CPUs), each with a 4.1-nanosecond clock, and 256 million 64-bit words of memory. Approximately 60 gigabytes of on-line, classified mass storage is available. A Cray Tape Controller, which provides IBM 3480 magnetic tape capability, and a Masstor M860 robotic storage device provide for off-line mass storage. The Cray–2, which runs strictly in a classified mode, is connected to the DSNET1 and various remote enclaves.

The Cray X–MP/48 has four CPUs, each with an 8.5-nanosecond clock, and approximately 40 gigabytes of unclassified mass storage. A 128-million-word solid-state disk is configured to use **ldcache** and as a primary swap device, thus making this resource available to all processes.

Both supercomputers run under UNICOS 6.1 and implement the Cray Data Migration procedures. Batch procedures are controlled by the network queuing system (NQS). A wide variety of output devices is available.

1.3 Applications Software

The ARLSCF software application codes and libraries are maintained by members of the ARLSCF staff. Brief descriptions of such software are contained in the chapter, "Applications Software." On-line assistance is provided via the **man** or **docview** commands. Requests for information or assistance should be sent by electronic mail to **craysupport@arl.army.mil**. Craysupport is an electronic mailbox read by numerous individuals, including scientists and hardware/software support personnel, and its use is encouraged for the exchange of questions, answers, and information.

1.4 Eligibility for User Accounts

Army organizations, other DoD organizations, and other Government organizations may use the ARLSCF. Contractors working for any of these organizations may use the computer facility; however, the sponsoring Government organization remains responsible for funding and management of the computer account.

Procedures for requesting accounts are contained in the chapter, "ARLSCF Computer Access."

1.5 Charges and Accounting

The charges for computing on the Crays at the ARLSCF are calculated solely on the amount of CPU time used, the priority category of the job, and the user's funding category. The charges are subject to annual review. The three priority categories are Normal, Express, and Deferred. The three funding categories are ARL mission funded and AMC Tech Base, Subscriptions, and Hourly. ARL and AMC computing that is neither mission funded nor Tech Base is charged as Subscription or Hourly.

Jobs submitted through NQS are in the Normal category unless they have been submitted explicitly to either the Express or Deferred queues. Interactive jobs run within the Normal category unless a user has specifically requested the Express or Deferred categories by use of the appropriate argument to the **–n** parameter of the **/usr/brl/bin/lim** command.

Specially scheduled dedicated sessions incur an additional charge for the transition into and out of the dedicated mode. The dedicated session is billed per wall clock hour including this transition time, and reflects the use of all CPUs on the machine.

See Appendix A for the current rate structure.

1.5.1 ARL Mission Funded and AMC Tech Base Users

The ARL and the various Army Research, Development and Engineering Centers (RDECs) comprising the Army Materiel Command (AMC) Technology Base have provided funding for the ARLSCF from their 6.1/6.2/6.3a funding categories. Users from these organizations working on projects funded similarly are not charged a dollar rate for their computing. Instead, each organization is allocated a certain number of hours of computing based on their share of the funds contributed. Jobs in the Normal category are charged the number of hours used; in the Express category, twice that much; and in the Deferred category, half that much. Dedicated sessions are charged quadruple the number of hours used, because all four CPUs are dedicated, including transition time. AMC Tech Base organizations whose users overspend the organization's allocation will be charged for the additional time at prevailing rates. See Appendix A for the current rate structure.

1.5.2 Subscriptions

Organizations which are neither ARL mission funded nor part of the AMC Technology Base but which anticipate high usage during the fiscal year may request a subscription to the ARLSCF. A subscription, which can be purchased for $300,000, $400,000, or $500,000, allocates to the subscriber a certain number of hours at a reduced rate. Jobs from subscribing organizations are charged against the allocation as if they were from AMC Tech Base organizations. Subscribing organizations whose users overspend the organization's allocation will be charged for the additional time at the subscription rate. There are no refunds or carryovers for a subscribing organization which underspends its allocation. See Appendix A for the current rate structure.

1.5.3 Monitoring Costs

The **charges** command is available on the Crays to enable users to keep track of their monthly or fiscal year usage. It reports the exact number of hours incurred in each priority category and in dedicated mode. The command **man charges** on either machine describes the various options. To determine his cumulative bill, an hourly customer multiplies the number of hours in each category by the hourly charging rate for that category and adds the results. Similarly, ARL mission funded, AMC Technology Base, and subscription customers multiply the number of hours in each category by the charging weight for that category, and add the results to get the cumulative number of hours decremented from their initial allocation.

1.5.4 Billing Questions

Questions regarding the charges and billing should be referred to the Billing Coordinator listed in Appendix B.

1.6 Classified Work

The ARLSCF operates the Cray–2 continually in a system high, periods processing mode. Access to the classified Cray–2 is via DSNET1, ARL (Aberdeen site) Secure Enclaves, and STUIII dialup. Individuals desiring this service should contact either the Secure Computer Access Team or the Information System Security Officer (ISSO) listed in Appendix B.

1.7 Schedules

For both Crays, a schedule of the normal operating times and scheduled deviations, and a history of unexpected down time due to power outages, machine crashes, etc., is kept in the file **/usr/pub/notd** on both Crays and on many minicomputers within the ARLSCF. Users interested in this information can display this file using the UNIX **cat**, **pg**, or **more** commands. In addition, on the Cray X–MP/48, the command **news notd** displays the current schedule.

1.8 Computer Security

Computer security at the ARLSCF is a serious and constant issue. The ARLSCF staff is vigilant in protecting against fraud, waste, abuse, and, in particular, unauthorized access. Users share in the responsibility for protecting these resources; failing to act responsibly will jeopardize a user's access to the ARLSCF. Some practices important to computer security are:

- Never write passwords where they can be seen by others — commit them to memory. Do not post passwords on terminals. Do not program passwords into terminals' function keys.
- Do not, under any circumstances, enter passwords into any file. In particular, the **qsub** command's **–u** option permits a password to be supplied as part of an NQS job. Do not use this option to provide a password.
- Do not share passwords and do not give them to others. If more than one person needs access to a specific project, the ARLSCF staff will establish multiple user accounts for that project.
- Do not include passwords in electronic mail messages.
- Do not leave terminals unattended and logged in.
- Report immediately any seeming discrepancy between reported computer usage (see the preceding section, "Monitoring Costs") and expected computer usage. Better to cry "wolf" and be wrong than to let an unauthorized user slip away.
- Report immediately any unrequested/unexplainable output, hard copy or on the terminal.
- Report immediately any moved, deleted, or altered files.
- Report immediately any strange, duplicate, or lost electronic mail.
- Report immediately any suspected security problem to the system administrator or the ARLSCF ISSO.

1.9 Responsibilities

The computing assets of the ARLSCF are precious resources, and the privilege of accessing them carries with it certain responsibilities. The ARLSCF staff is responsible for maintaining and operating the various

computer resources. Users share in the responsibility for maintaining and operating this facility and are responsible specifically for:

- Reading and complying with the various guidelines and standard operating procedures (SOPs).
- Monitoring computer usage to prevent waste and fraud.
- Notifying the ARLSCF immediately of any change in a user's status.
- Ensuring that passwords are not compromised, that equipment is always physically secure, and that access to equipment is always controlled.
- Using the ARLSCF resources for official business only.
- Creating backup files for their own important work.
- Reading system bulletins to keep current with changes to the system (**news news** tells how).
- Arranging for the disposition of tape, disk, and tape cartridge files upon termination of an account.
- Limiting one's concurrent NQS jobs to two per queue.
- Limiting one's interactive jobs so as not to hog the system.
- Removing or archiving unneeded files.
- Limiting the size of files/messages mailed from the supercomputers.
- Giving permissions only to the owner for the . (dot) files in one's user space.

1.10 Computational Support Branch

The Computational Support Branch handles all output, maintains the reference and tape storage libraries, handles tape processing and file restoration, and provides users with necessary documentation. Questions of a general nature or to ascertain the responsible individual should be directed to the Chief (see Appendix B for names).

1.11 User Services

The ARLSCF Scientific Support Team serves as the User Services/Help Center group. The team is dedicated to helping users utilize the ARLSCF facilities to their fullest extent. The ARLSCF staff coordinates training, provides in-house training, consults with users on any problems they may encounter, and assists in porting codes to the supercomputers.

In addition to those people listed in Appendix B, support is obtained from the entire ARL technical community. Questions concerning procedures, implementation issues, and virtually any question relating to high performance computing should be addressed electronically to **craysupport@arl.army.mil**.

Electronic mail is the preferred means of communication within the ARL. The craysupport mail list is monitored by members of the ARLSCF staff responsible for operations, members of the on-site Cray Research Inc. staff, and numerous members of the ARL scientific research staff. Any user desiring to become a craysupport recipient may request to be included on the mailing list. Please send your request to **support–request@arl.army.mil**.

The ARLSCF staff communicates hardware and software changes, explanations for extended down time, training class and seminar announcements, and other matters to the user community through the info–cray mail list. This mail list includes everyone receiving craysupport mail plus others who only want to receive informative messages about the Cray. Users desiring to be on this mail list but not on craysupport should send their request to **info–cray–request@arl.army.mil**.

1.12 Eastern Time

The ARLSCF uses Eastern Time, either Eastern Standard Time or Eastern Daylight Time.

2. ARLSCF Computer Access

The Army Research Laboratory SuperComputer Facility (ARLSCF) is located at Aberdeen Proving Ground, MD. It provides services to Army scientists and engineers on site and around the country. Access to the ARLSCF computers, for most users, usually involves a connection via a computer network or telephone line. This chapter describes the procedures to request accounts on ARLSCF computers, and to connect with and log into and out of them.

2.1 Requesting an Account

Persons who are employees of Government agencies, civilian contractors working for Government agencies, or employees of firms which are working under Government contracts may request accounts on ARLSCF computers. For Cray–2 accounts, a security clearance is required. To apply for accounts, send a written or electronic mail request to the appropriate address listed in Appendix B. The request should specify the names of all persons desiring accounts, their phone numbers, agency addresses, supervisors, and the computers to which access is desired (including ARLSCF front end machines, if necessary). Account request and security clearance forms, as appropriate, will be sent to each individual upon receipt of requests. User id's, accounting classification numbers, and dialup phone numbers are issued upon receipt and approval of these forms. This procedure normally takes about 1 week to complete, and is necessary to ensure that the ARLSCF abides by the U. S. Government policy regarding access to supercomputers. Supervisors of ARL or Army Materiel Systems Analysis Activity (AMSAA) employees may contact the Cray Account Administrator in person to establish accounts for them.

2.2 Methods of Access

The ARLSCF computers are accessible through the INTERNET (including MILnet, ARPAnet, and NSFnet), through other computers on the ARL-APG site LAN (BRLnet), or through the classified network. A direct dialup phone connection cannot be used directly to access the supercomputers; however, such connections can be established to a minicomputer on the BRLnet.

2.2.1 INTERNET Access

The INTERNET is a collection of networks including the NSFnet, the ARPAnet, and the MILnet (among others), all three of which use the TCP/IP communications protocol. These three networks are managed and supported by the DoD and constitute the Defense Data Network (DDN).

Users with INTERNET connectivity and running TCP/IP software may access the ARLSCF computers by invoking the TELNET utility:

telnet *internet_target_host_name*

or

telnet *internet_target_numerical_address*

Upon establishing a connection, the user receives a login prompt.

Users with TAC (Terminal Access Controller) access may use the TAC to access the ARLSCF computers. Once a connection to the TAC has been established, it may be necessary to enter **CTRL–Q** to wake up the TAC. At that point, enter

@o[pen] *internet_target_numerical_address*

The TAC will then prompt for and verify the user's TAC id and access code before connecting to the target machine. After logging out of the target machine, the user will be returned to the TAC. Enter

@c[lose]
@l[ogout]

to close the connection (may not be necessary) and log out of the TAC. On most TACs it is possible to reach the DDN Network Information Center by entering

@n[ews]

The INTERNET names and addresses of the Cray X–MP/48 and of a front end computer are:

Cray X–MP/48	**patton.arl.army.mil**	**128.63.23.5** or **192.5.21.20**
Gould 9080	**adm.arl.army.mil**	**192.5.25.4** or **192.5.21.30**

The Cray–2 is on the DSNET1, a classified network, rather than the INTERNET. Connections can be made in a similar manner, but neither the names nor addresses can be provided because they are classified. Off-site Cray–2 users who need assistance in establishing their connection mode to the machine can contact the Secure Computer Access Team (**scat@arl.army.mil**) for help.

2.2.2 The BRLnet

The ARL-APG LAN (BRLnet) provides direct access to both Cray supercomputers through front end machines. The BRLnet is available to users with modems via several dialup phone lines (300, 900, 1200, 2400, 4800, and 9600 baud), and to users with direct connections to computers on the BRLnet. The dialup numbers and the baud rates for each will be released to the user once an account has been approved.

There are two methods by which ARLSCF computers may be accessed from the BRLnet. The first is through use of the **rlogin** utility (e.g., **rlogin patton.arl.army.mil** for the Cray X–MP/48). The second is by way of the TELNET utility, as described in the preceding section, "INTERNET Access."

2.3 UNICOS Login Message

ARLSCF computers operate under UNIX, or variants thereof (e.g., UNICOS). Once a connection has been established, the user receives a warning regarding unauthorized access and then the **login:** prompt. The proper response is to enter the assigned username followed by a carriage return, after which the **Password:** prompt is issued. The user responds with the password followed by a carriage return. Once login has been completed, the system displays the date and time of the last login followed by the message of the day, a short bulletin containing system information of which the user should be aware during the login session.

2.4 Logging Out

To log out of an ARLSCF computer, enter a **CTRL–D** as the first character on a line or enter the command **logout**. If the ARLSCF computer has been accessed from another computer, the user is returned to that computer and should follow the normal procedure for logging out of that computer. If the connection

has been made through the TAC, then the user is returned to the TAC and should follow the normal procedure for logging out of the TAC, as described in the preceding section, "INTERNET Access."

Figure 2.1 displays a sample login session demonstrating access to the ARLSCF Cray X–MP/48 via the TELNET utility using the INTERNET numerical address. Actual keyboard entries are emboldened, • indicates a RETURN, and ^ indicates a **CTRL**–.

*****

```
% telnet 128.63.23.5•
Trying 128.63.23.5...
Connected to 128.63.23.5.
Escape character is '^]'.


Cray UNICOS (patton) (ttyp011)


**************************************************************************
*       Unauthorized users of this computer are subject to prosecution.  *
**************************************************************************

login: user_name•
Password: user enters password, not echoed to screen•
Last successful login was : Fri Nov 6 09:00:01 from adm.arl.army.mil
**************************************************************************
*       Running UNICOS 6.1.  Report any problems to craysupport@arl.army.mil*
*                                                                        *
*       Use news to get info about the system.  "news news" tells how.   *
*       Any unread news items are denoted by "news: filename" upon login.  *
**************************************************************************
news: notd cos change_passwd gc schedule at-batch-cron deferred.q nqs.queues
 abaqus nqs.policy disspla.imagen restorations tmpdir express.que 6.1upgrade
 arlscf_policy news flint sysadmin

patton> date•
Fri Nov 6 10:30:15 EST 1992
patton> ^DConnection closed by foreign host.
%
```

Figure 2.1. TELNET Access to the ARLSCF Cray X–MP/48

*****

2.5 ARLSCF Points of Contact

See Appendix B for the current list of points of contact for various problems or questions that may arise regarding the ARLSCF.

Intentionally Left Blank

3. UNICOS – The Operating System

This chapter applies specifically to the ARLSCF Cray computers; however, to the considerable degree that UNICOS is similar to AT&T System V UNIX or BSD UNIX version 4.2, the information presented applies also to the various other computers at ARLSCF.

3.1 UNICOS Features

UNICOS is the UNIX-based† operating system running on the Cray X–MP and the Cray–2 supercomputers at the ARLSCF. It consists of the core operating system, an extensive set of utilities, and several command line interpreters, known as shells. The flexibility and portability of the UNIX operating system make it an ideal choice for supercomputer operations. The Cray enhancements incorporated in UNICOS are largely in these areas:

- File system improvements to provide supercomputer class I/O performance.
- Support for multiple processors and multitasking.
- Additional debugging aids.
- System accounting features.
- Additional batch processing capabilities.
- Increased CPU time and memory size limits.

3.1.1 On-Line Help

UNICOS provides an on-line help facility. Executing **man** *command_name* displays the command's reference manual pages on the screen. For example, **man man** provides detailed information on the operation of the **man** command. Additional information may be found in the chapter, "On-line Information."

3.1.2 Shells

A shell is a user's interface to the UNICOS operating system. As commands are entered, the user's shell interprets the line and relays the information to the operating system for processing. A command programming language is provided, and executable files, known as shell scripts, can be produced. Shell scripts are used to group shell commands so they may be executed conveniently and repeatedly. UNICOS supports the C shell, **/bin/csh**, the Bourne shell, **/bin/sh**, and the Korn shell, **/bin/ksh**. Users are not limited to these three shells because UNICOS permits implementation of others. Two of ARLSCF's own shells, the TC shell, **/usr/brl/bin/tcsh**, and the TB shell, **/usr/brl/bin/tbsh**, are available to users.

† AT&T System V UNIX with BSD enhancements and Cray enhancements.

All five of these shells are similar, but each has its own unique capabilities and features. Users, initially assigned the Bourne shell, may change their shell as desired by executing
chsh *full_path_name_of_desired_shell.*

3.1.2.1 Command Syntax

In general, shell commands follow a common syntax:

command_name [*–option* [*argument*] [*–option* [*argument*] ...]] *operands*

The command and its arguments are separated by spaces. Each command line is terminated by a **return**.

Most commands allow certain minor abbreviations in syntax (e.g., multiple *option*s without *argument*s concatenated into one string prefixed with a single hyphen), and certain commands do not strictly follow it. Detailed information about any command may be obtained on-line by executing **man** *command_name.*

3.1.2.2 Metacharacters

Metacharacters are keyboard characters which have special meaning to UNICOS, to the user's shell, or to a UNICOS program. The UNICOS metacharacters, whose meanings may be context dependent, and for which complete descriptions may be obtained by executing **man sh**, are:

. : ; ? ! ( ) [ ] { } | ^ < > & * # $ @ \ ' ` " newline space tab

In addition, many UNICOS programs (in particular, editors) recognize their own metacharacters, usually having considerable overlap with the UNICOS metacharacters. For detailed information on such programs, execute **man** *command_name.*

3.1.2.3 A Few Useful Commands

UNICOS provides a large suite of utilities for creating, deleting, inquiring about, and otherwise manipulating files. A few of the more basic ones, to get the complete novice started, follow. These commands accept a number of options, arguments, and operands, for which detailed information may be obtained by executing **man** *command.*

cat *files* [>[>] *other_file*] Concatenate *files* into *other_file*, which is created or overwritten (>), or created or appended to (>>). *files* is a blank-separated list of file names. Without *other_file*, the concatenation is to standard out.

cd [**name**] From the current directory, move into directory *name*; *name* must provide sufficient identifying information. With no argument, return to the user's home directory.

cp *file1 file2* Copy *file1* onto *file2*, overwriting any existing *file2*.

diff *file1 file2* Perform a line by line comparison of *file1* and *file2*, displaying differences at the terminal. **diff** *file1 file2* > *file3* redirects the results into *file3*.

jove *file* Enter **jove**, a popular and well supported screen editor at ARLSCF, creating a copy of *file* as the work file. Additional information may be found in the chapter, "Text Editors."

ls –al List in long form (**–l**) all (**–a**) files in the current (sub)directory.

mkdir *name* Create a new sudbirectory, *name*.

mv *file1 file2* Move (actually rename, subject to certain conditions) *file1* to *file2*, destroying any existing *file2*.

pwd Display the complete name of the current (i.e., present working) directory.

rm *files* Remove all *files* in the blank-separated list of *files*. Be exceptionally careful of any variant of this command using metacharacters, to avoid removing more files than is intended.

rmdir *name* Remove the empty subdirectory *name*.

3.1.3 Network Queuing System

The NQS is the only way to submit Cray jobs requesting more than 10 minutes execution time. In addition, NQS submissions are advantageous even for shorter jobs because

- NQS jobs lend themselves to management by ARLSCF staff, alleviating certain problems which can occur when too many jobs request too much memory.
- NQS jobs do not retain control of the user's shell/terminal after submission.
- NQS jobs are automatically checkpointed before intentional system shutdowns, thus allowing recovery when the machine returns.

Additional information may be found in the chapter, "Batch Jobs."

3.1.4 File Migration

Disk space on the ARLSCF Crays, although large, is limited. File migration is a system process whereby larger, less often used disk files are automatically, and more or less transparently to the user, transferred to tape storage. Additional information may be found in the chapter, "File Storage – On- and Off-Line."

3.2 The File System

The general characteristics of the UNICOS file system are typical of many UNIX systems. It is a hierarchical tree structure consisting of several types of files. Individual files may contain text, data, or binary information. The files are grouped into manageably sized file (sub)systems to optimize disk space and to provide effective I/O throughput. Each user's set of files is organized as a branch of the tree, whereon each one of that user's files and subdirectories is a leaf or a sub-branch. To create, delete, modify, or execute files, a user must have the appropriate access permissions to those files. Information about permissions may be obtained by executing **man chmod** and **man ls**.

3.2.1 Types of Files

The UNICOS file system incorporates several different types of files, of which the following are the more likely to be of interest to typical users:

regular The typical user file, containing text, data, source code (a form of text), object code, and executable programs.

directory A file containing information which enables the operating system to access other files (which may include other directories) within the directory, and to return to the directory's parent. (A file's or directory's parent is the branch from which it sprouts.)

migrated A regular file which, due to demands for disk space, has been "migrated" to off-line storage. Only the name of the file and information sufficient to retrieve it in a more or less transparent way remains on-line.

special Typically, a file containing information enabling the operating system to deal with devices used for input and output. The file actually represents the device. On other UNIX systems, certain of these files are directly addressed by users; for example, to read a "tar" tape, a user might execute **tar /dev/rmt0**, **/dev/rmt0** being a special file representing a tape drive. Under UNICOS, however, other methods are used (see the **tar** paragraph in the chapter, "File Transfers," and users have little need to explicitly name special files. The special file **/dev/null** is a notable exception. Output redirected to **/dev/null** (**>/dev/null**) on UNICOS and other UNIX systems is effectively suppressed.

3.2.2 Directories and Paths

From the operating system's perspective, a directory is a file containing information which enables it to access other files (which may include other directories) within the directory, and to return to the directory's parent. From a user's perspective, a directory is a hierarchical structure of files and (sub)directories. A user's file structure begins with the user's home directory, into which he is placed upon logging in, and in which he may create as many files and nested or unnested (sub)directories as desired, in a tree structure which extends as broadly and as deeply as desired.

The origin of the entire UNICOS file system is the "root" directory, with complete name "/". Some few of the topmost levels in root are:

/	root; the directory containing everything
/bin	directory containing user-oriented commands
/bin/cat	the **cat** command
/bin/ls	the **ls** command, AT&T System V style
/etc	directory containing data and commands related to system administration
/usr	directory containing system files and directories
/usr/bin	directory containing user-oriented commands
/usr/ucb	directory containing user-oriented commands from University of California at Berkeley
/usr/ucb/ls	the **ls** command, Berkeley style
/usr/brl/bin	directory containing user-oriented commands developed at the then USABRL
/usr/local/bin	directory containing user-oriented commands not part of the normal UNICOS distribution and of somewhat specialized interest; often proprietary
/dev	directory containing special files for I/O devices
/lib	directory containing system libraries
/organization_name	directory containing home directories of that organization's employees

This brief list raises questions about navigating within and across directories, and accessing specific files. The names listed above are, in every case, full path names — that is, they specify, unambiguously and absolutely (i.e., with respect to root, the origin of the entire file system), the location in the file system tree of the named file.

A user can reference a file in two completely different ways: as a command to be executed (e.g., the **cat** in **cat** *file*), or as an operand (e.g., the *file* in **cat** *file*).

First, we discuss files as operands. A file name (e.g., **mydata**, **mypgm.f**, or **a.out**) is a relative path name referencing a file of that name in (relative to) the current directory. A file name prefixed by one or more subdirectory names (e.g., **pressure/oct91** or **temperature/internal/jan92**) but not with an initial slash, is a relative path name referencing a file whose name is the last component in the string, and which is to be found by descending the directory hierarchy as indicated by the successive component

names, starting in the current directory. Note that the components of the path are always separated by single slashes. A prefix of "./" on a relative path name is equivalent to prefixing it with the current directory name, which is merely redundant in this discussion, but will not be later. A prefix of "../" on a relative path name is the logical equivalent of prefixing it with the parent of the current directory. A ".." appearing as an intermediate component in a relative path name is the logical equivalent of the name of the parent of the directory immediately before the "..". Note that these two effects of ".." cannot be duplicated with actual path names because ".." is the only symbol which causes the path to travel back up the directory tree. Starting at root is not the equivalent because it forces initiation at the origin, not travel in the reverse direction.

Now consider file names used as commands, e.g., **ls** to obtain a list of file names in the current directory. Everything stated concerning file names used as operands applies here, and more. Rarely is a user's current directory the directory which contains a UNICOS command such as **ls**. Hence, a user almost always would be required to enter a full path name (e.g., **/bin/ls** or **/usr/ucb/ls**) for a command, but for the **PATH** variable. Each user has a **PATH** variable, which is assigned a default value when the user is first granted an account. Users may change their own **PATH** variable definitions to suit their own needs. The techniques for setting the variable and the considerations involved in choosing its value are beyond the scope of this short discussion. Users unfamiliar with the significance of alterations to their **PATH**s are urged to seek competent advice before attempting alterations.

A user's **PATH** contains a colon-separated list of full path names to the directories within which are found the commands of which that user makes use. The names are in a particular order. When a user issues a command by entering its name without explicit path information, UNICOS does not expect to find that command in the current directory. Rather, UNICOS searches all the directories whose full path names are in the user's **PATH** variable, in the order in which they occur, until an executable file whose name matches the command name is found. That file is the command which is executed. Thus, for example, by incorporating **/bin**, or **/usr/ucb**, or both in the desired order, in their **PATH**s, users may control which **ls** command is executed when they enter **ls** as a command. The "other" **ls** still may be selected by entering its full path name as a command when desired. A user might even have his own tailor made version of **ls** in one of his own directories (typically named **.utility** or **.bin**) and name that directory early in his **PATH**. Whenever a command name is entered with explicit full or relative path name information, **PATH** is not used for the search; rather, the rules for finding a file named as an operand are followed. The usefulness of "./" as a command name prefix (merely redundant as an operand name prefix) is that, when a command is so prefixed, it has explicit path information. Thus, *./file* means execute *file* found in the current directory, rather than *file* found by searching the **PATH**.

The different performance provided by different commands with the same names (e.g., **/bin/ls** versus **/usr/ucb/ls**) can be surprising to inexperienced users. Sometimes it can be frustrating. Consider the **rsh** command. **/usr/ucb/rsh** opens a shell on a remote machine. It permits a user logged in to an ARLSCF Cray to execute a command on some remote machine networked with the Cray. **/bin/rsh**, however, opens a restricted shell on the Cray. A user who expects to execute **/usr/ucb/rsh** when he types **rsh**, but instead executes **/bin/rsh**, is likely to become frustrated unless he understands the selection mechanism. For the convenience of users who give priority to **/bin** over **/usr/ucb** in their **PATH**s, UNICOS provides **/usr/ucb/remsh**, a synonym for **/usr/ucb/rsh**. **remsh** cannot result in executing **/bin/rsh**, regardless of a user's **PATH**.

3.2.3 File Names

UNICOS file names may contain up to 14 characters and consist of letters, numbers, periods, underscores, and other printable and unprintable characters. Upper and lower case letters are different characters. It is most strongly emphasized that users ought to limit their files' names to those consisting only of letters, numbers, periods, and underscores. To use characters in addition to these is to court unexpected, and sometimes disastrous, behavior.

By convention, certain file name suffixes have special meanings under UNICOS. Various UNIX systems require various ones of these suffixes for various commands. For example, **cft77** accepts but does not require the **.f** suffix whereas **cf77** requires it.

- **.a** file contains an object code library
- **.c** file contains C source code
- **.f** file contains Fortran source code
- **.F** file contains Fortran source code to be preprocessed by **gpp** under **cf77**
- **.h** file contains C source code header information
- **.l** file contains listing output from a compiler or assembler
- **.o** file contains object code
- **.p** file contains Pascal source code
- **.s** file contains assembler source code
- **.z** file has been **pack**ed; **unpack** will return it to its original condition
- **.Z** file has been **compress**ed; **uncompress** will return it to its original condition

3.2.4 File Permissions

Each file (including directories) has associated permissions which control access to the file. A file inherits its set of permissions from its owner when the file is created. The owner may change the permissions of existing files (**chmod**), or change the permissions which all his files acquire automatically upon creation (**umask**). When a user is initially granted an account, the "automatically acquired upon file creation" permission set is defined so that no one other than that user has any access to his files. Current policy at ARLSCF is that users who increase the accessibility of their files assume responsibility for any security violations which occur due to that increased accessibility. Be aware that ARLSCF computers are linked to international networks and that there have been attempts to gain unauthorized access.

ls –**l** displays, among other things, the permissions associated with files. For example, **ls** –**l** *file* might yield

–rwx–––––– (other information) *file*

The leftmost 10 columns are of interest. The first column indicates, not permission, but that *file* is a regular file. A **d** in this position would indicate a directory (ultimately, a directory is a file, but a specialized one whose contents enable the system to locate the other files "in" the directory), an **m**, a migrated file, and there are a few other characters which might appear. The next nine columns are to be considered as three triplets, the first triplet pertaining to *file*'s owner, the second, to *file*'s group, and the third, to everyone else. The permissions possible are to **r**ead from, to **w**rite to, and to **e**xecute as a program the already existing file. In this example, *file*'s owner can do all three, and no one else can do anything. For a particularly precious data file, the owner might choose permissions **r**––––––––. There are a few other characters which might appear instead of **r**, **w**, and **x**.

Permissions on a directory have a somewhat different interpretation. To have write permission to a directory is to be able to modify the directory file itself — that is, to be able to create and remove files within the directory. To be able to manipulate files conveniently, a user needs both read and execute permission on his parent directory. Read permission without execute permission on a directory allows a user to do nothing more than obtain the names of the files therein. Execute permission without read permission on a directory allows a user to do everything he could do with both, except obtain the names of the files therein. Therefore, without read permission, the user must have independent knowledge of the file names in order to access the file.

4. X11 Window System

Various computers at the ARLSCF, including both Crays, provide version 11 of the X Window System, henceforth referenced as X11.

X11 programs and libraries provided by Cray Research, Inc. are in **/usr/bin/X11** and **/usr/lib**, respectively. Detailed information may be found in the Cray publication, *UNICOS X Window System Reference Manual*, SR–2101 6.0. There are X11 interfaces to the Cray debugger, **cdbx** (discussed in the Cray publications, *UNICOS CDBX Debugger User's Guide*, SG–2094 6.0, and *UNICOS CDBX Symbolic Debugger Reference Manual*, SR–2091 6.1) and to the Cray performance utilities **atexpert**, **atscope**, **flowview**, **perfview**, and **profview** (discussed in the Cray publication, *UNICOS Performance Utilities Reference Manual*, SR–2040 6.0).

X11 programs and libraries from other sources are in **/usr/X11/bin** and **/usr/X11/lib**, respectively.

4.1 ARLSCF X11 Programs

Some of the X11 programs available at the ARLSCF are listed below. Additional information may be found in the **man** pages. Depending on a user's environment, it may be necessary to augment the **man** command with the argument **–M /usr/X11/man** or to add **:/usr/X11/man** to the environmental variable, **MANPATH**. All these programs open windows via **xterm**. The mail- and news-related programs are not available on the Crays.

xchoose provides a menu capability for shell scripts. Accepts the menu items as command line arguments, or as lines from standard input, and writes to standard output the menu item selected. Usage: **CHOICE=`ls | xchoose`**.

xdu displays graphically the output of **du**. Usage: **du | xdu**.

xftp X11 version of **ftp**.

xmsg reads electronic mail of the form used at the ARLSCF, and is patterned after **msg** and **xrn**, with certain extensions. Opens multiple windows to read, send, answer, and forward mail. The default mail file is **$HOME/mailbox**.

xod displays binary files created on various machines. There is a menu selection of the creating machine (Convex, Cray, IBM, IEEE, Vax) and of the desired display conversion.

xrcvalert mail receipt notification program whose invocation is specified in the user's **.maildelivery** file. When mail is received, the mail system invokes **xrcvalert**, which prints a line in the root window screen and sounds a bell.

xrn the X11 counterpart of **rn**.

xrgbsel displays on a color X display those colors which can be specified by name in an X program. It reads file **/usr/X11/lib/X11/rgb.txt** to obtain color names and mixes.

xtw displays a directory tree. The current directory's ancestry is shown as a trunk to its left, and its descendants expand into a tree to the right. The tree may be walked with a mouse, and the display expands or contracts as appropriate.

xuptime prints the time, the length of time the system has been up, the number of users logged in, and the average number of jobs in the run queue over the last 1, 5, and 15 minutes, and plots a graph of the CPU load verses time. Usage: **uptime | xuptime**.

4.2 Compiling X11 Programs

X11 C programmers are strongly encouraged to create an **Imakefile** and use command **xmkmf** to to generate a **makefile** with the proper definitions for a given computer. If the **makefile** is created manually, it is likely that the arguments **–DX_NOT_POSIX** and **–DX_NOT_STDC_ENV** will be needed on the **cc** command line.

4.3 ARLSCF Fortran Interface to X11

/usr/X11/lib/libXf77.a is the ARLSCF Fortran X11 library. It contains subroutines of two kinds, one kind to provide a menu capability for Fortran programs, and the other, to create plots from Fortran programs.

4.3.1 X11 Menus from Fortran

The **xlmenu** subroutine causes a menu to pop up on the screen, centered on the current position of the mouse cursor, and returns the index of the menu item selected upon release of the mouse button. Useful for prompting an interactive Fortran program's user to select an item from a list. **XMENU** is the same as **XLMENU**, except that there is no title. The font used within the menus is specified in the user's X application resources data base file, e.g., **Xmenu.font: 8x13bold**. Typically, this file is named **.Xdefaults** or **.Xres**. For example,

CALL XLMENU (*title,list,nlist,jpickd*)

title	a character string, the title of the menu
list	a character array, each element being a menu item
nlist	an integer, the number of menu items in *list*
jpickd	an integer, the returned ordinal of the selected menu item

4.3.2 X11 Plotting from Fortran

All arguments beginning with **i** are **INTEGER**, all others, **CHARACTER**. All heights, widths, distances, and positions are expressed in pixels, relative to (0,0) at the lower left corner of the window.

XCOLOR (*string*)	Sets the current foreground color to that specified by *string*. A list of strings with their red, green, and blue components is located in **/usr/X11/lib/X11/rgb.txt**. Command **xrgbsel** displays the colors specified in **/usr/X11/lib/X11/rgb.txt**.
XDRAW (*ix,iy*)	Draws a line to *ix,iy*, which becomes the current position.
XERASE	Erases the window.
XFLUSH	Flushes output to the X display.
XFONT (*string*)	Sets the current font as specified by *string*. Command **xlsfonts** lists, and commands **xfontsel** and **xfbrows** display, the various fonts available on the display device.
XINIT	Required. Must be called first to create the window in accord with the foreground, background, and geometry resources defined in the user's X application resources data base file (typically named **.Xdefaults** or **.Xres**), or in accord with defaults. For example,

Xplot.forground: yellow
Xplot.background: blue
Xplot.geometry: 600x450+200+100

The defaults are black, white, and half the screen width by half the screen height, with no positioning, respectively.

XINQ (*inq*) provides information about the X11 server. *inq* is an 8-element integer array in which are returned:

inq(**1**)	window width
inq(**2**)	window height
inq(**3**)	screen width
inq(**4**)	screen height
inq(**5**)	pixels per inch horizontally
inq(**6**)	pixels per inch vertically
inq(**7**)	character height
inq(**8**)	maximum number of colors

XLOCAT (*ix,iy,ibuttn*) Returns pixel location *ix,iy* pointed to by the mouse when button *ibuttn* is pushed. The mouse buttons are usually numbered left to right.

XLTYPE (*string*) Sets the line type to *string*, which must be one of **solid**, **dot**, **dash**, and **dotdash**.

XLWID (*iw*) Sets the line width to *iw* pixels.

XMOVE (*ix,iy*) Moves the pen to *ix,iy*, which becomes the current position.

XPAUSE Pause until a mouse button is pushed in the plot window.

XPOLYF (*ix,iy,np*) Draws a closed filled convex polygon using the current foreground color. Integer arrays *ix* and *iy* contain *np* coordinate pairs. If the first and last points do not coincide, the polygon is automatically closed.

XRGB (*ired,igreen,iblue*) Sets the current foreground color to that specified by the arguments, which have values from 0 through 32767, and specify the intensity of the corresponding primary color.

XRSIZE (*iw,ih*) Changes the width and height of the window to *iw,ih*, respectively. Should not be called after plotting has begun but may be called between plots.

XRSTR (*ix,iy,iang,string*) Places a rotated *string* at position *ix,iy* with angle of rotation *iang*. *iang* increases in the counterclockwise direction with zero at the 3 o'clock position, and is in degrees times 64.

XSTR (*ix,iy,string*) Places the lower left corner of *string* at position *ix,iy* on the plot, which position becomes the current position.

XTEXTW (*string,iw*) Returns in *iw* the width in pixels of *string*.

XTITLE (*title*) Changes the title on the window and on the window manager icon to *title*.

4.3.3 Sample Fortran Program with X11 Interface

Figure 4.1 presents a simple Cray Fortran program which computes the monthly outstanding balance of a simple interest loan. Results are displayed graphically with two different formats. The **segldr** command requires arguments **–L/usr/X11/lib –lXf77 –lX11**.

*****

```
      program loan

c     affects appearance; eg, 3.0 means 3000. is by 1000s
c                             and 2999. is by 100s
c                             5.0 means break is at 500.,
c                             5000., 50000., etc.
      parameter (break=5.)

      character abc*10, format*26, title*63
      dimension npix(8), iscr(2), ires(2)
      equivalence (iscr(1),npix(3)), (ires(1),npix(5)),
     +            (ichht,npix(7)), (ncolor,npix(8))
      dimension date(600), bal(600)
      iwide(x) = int (log10(x+.01))+1

c     read initial amount, monthly payment, annual interest percent
      write (*,'(/2a/)')
     +     'Enter initial amount, monthly payment, annual percent,',
     +     ' with decimal points:'
      read (*,*) bal(1), pay, fint
      date(1) = 0.

c     create the format to create the title
      write (format,'(a,i2.2,a,i2.2,a,i1,a)')
     +     '(a,f',iwide(bal(1))+3,'.2,a,f',
     +     iwide(pay)+3,          '.2,a,f',
     +     iwide(fint)+3,         '.2,a)'
c     create the title
      write (title,format)
     +     'LOAN AMORTIZATION:  $',bal(1),' by $',pay,' monthly, at ',
     +     fint,'%'

c     compute monthly balances
      fint = 1. + fint / 100. / 12.
      n = 1
      do while (bal(n).gt.0.)
         n = n + 1
         date(n) = date(n-1) + 1./12.
         bal(n) = bal(n-1) * fint - pay
      enddo
      bal(n) = 0.

      call xinit
      call xtitle (title)
      call xfont ('screen.b.14')
c     border width in pixels
      ibdr = 10
c     distance between vertical axis tic marks, user units
      yinc = 10. ** int (log10(bal(1)/break+.01))
c     plotting limits, user units
      xmin = date(1)
      xmax = aint (date(n)+1.)
      iymax = int (bal(1)/yinc+1.)
      if (mod(iymax,2).ne.0) iymax = iymax + 1
      ymax = iymax * yinc

c     horizontal space for vertical axis labels, pixels
```

```
      abc = '1000000000'
      call xtextw (abc(1:int(log10(ymax+.01))+1),ix)
      ix = ix + ibdr

c     output results, change window size, output results
      do 100 loop = 1 , 2
         if (loop.eq.2) call xrsize (800,600)
c        query the X server
         call xinq (npix)

         iy = ichht * 2 + ibdr
         mx = npix(1) – ibdr*2
         my = npix(2) – ibdr
         xscl = real (mx–ix) / (xmax–xmin)
         yscl = real (my–iy) / ymax
         call xtextw ('Payment Years',iix)
         call xstr (ix+(mx–ix–iix)/2,ibdr,'Payment Years')

c        draw axes with solid lines
         call xltype ('solid')
         call xmove (ix,iy)
         call xdraw (ix,my)
         call xdraw (mx,my)
         call xdraw (mx,iy)
         call xdraw (ix,iy)

c        draw results – red lines if color is available
         if (ncolor.gt.2) call xcolor ('red')
         call xmove (ix,int(bal(1)*yscl+iy))
         do i = 2 , n
            jx = (date(i)–xmin) * xscl + ix
            jy = bal(i) * yscl + iy
            call xdraw (jx,jy)
         enddo

c        change background to black if color is available
         if (ncolor.gt.2) call xcolor ('black')

c        draw and label grid – use dotted lines
         call xltype ('dot')
c        vertical lines
         write (abc,'(i2)') int (mod(xmin,100.))
         call xtextw (abc(1:2),jjw)
         call xstr (ix–jjw/2,iy–ichht,abc(1:2))
         do id = nint(xmin)+1 , nint(xmax)–1
            jx = (real(id)–xmin) * xscl + ix
            call xmove (jx,iy)
            call xdraw (jx,my)
            write (abc,'(i2)') int (mod(real(id),100.))
            call xstr (jx–jjw/2,iy–ichht,abc(1:2))
         enddo
         write (abc,'(i2)') int (mod(xmax,100.))
         call xstr (int((xmax–xmin)*xscl)+ix–jjw/2,iy–ichht,abc(1:2))
c        horizontal lines
         write (abc,'(i10)') 0
         call xstr (ibdr/2,iy–ichht/3,abc(11–iwide(ymax):10))
         do id = 2 , iymax–2 , 2
            jy = real(id) * yinc * yscl + iy
```

```
            call xmove (ix,jy)
            call xdraw (mx,jy)
            write (abc,'(i10)') nint(id*yinc)
            call xstr (ibdr/2,jy-ichht/3,abc(11-iwide(ymax):10))
         enddo
            write (abc,'(i10)') nint (iymax*yinc)
            call xstr (ibdr/2,iymax*int(yinc*yscl)+iy-ichht/3,
     +                 abc(11-iwide(ymax):10))
         call xpause
         call xerase
  100 continue
      end
```

Figure 4.1. Fortran Program Producing X Window Graphical Output

*****

4.4 X11 Interface to PVI

To use the X11 PVI driver on a remote system such as the Cray X–MP/48, patton.arl.army.mil, **/usr/brl/bin/tcsh** users and **/bin/csh** users must

setenv PVI_DEV_1 x11
setenv DISPLAY *display_name.domain_name*:**0**

and **/bin/sh** users must

PVI_DEV_1=x11
DISPLAY=*display_name.domain_name*:**0**
export PVI_DEV_1 DISPLAY

A typical domain name for ARL users is **arl.army.mil**. The trailing **:0** indicates that the display consists of only one device.

In addition, on an X terminal or workstation, a user would execute **xhost** *host_name.domain_name* or would add (or request to have added) a *host_name.domain_name* line to file **/etc/X0.hosts**. The **xhost** command must be executed each time the host is brought up under X Windows, whereas the entry in **/etc/X0.hosts** need be made only once.

The following entries in the X resource data base manager file (typically **.Xdefaults** or **.Xres**; read by **xrdb** during the initiation process for running X windows) explicitly set a user's PVI display to the defaults for a 1152x900 X display. The user may modify those characteristic as desired.

PVI.foreground:white
PVI.background:black
PVI.geometry:576x450

5. On-Line Information

This chapter provides an introduction to the ARLSCF on-line information and help sources. Boldface indicates information to be taken literally or, in a representation of screen output, something typed in by the user. Italics indicate something to be replaced with actual values by the user, operating system, or software in use; for example, *prompt*> denotes the system prompt and indicates that *prompt* will be replaced by the given system's name or by some other prompt specified by the user.

5.1 Manual Pages

All computers at ARLSCF have on-line copies of the various UNIX user manuals, referred to as "man pages." They provide information about system utilities and user commands. To view a man page, enter **man** *command.*

For example, Figure 5.1 is an extracts of the information obtained in response to entering **man jove**. Note that output on the various ARLSCF computers may differ slightly in format and even content.

*****

JOVE(1B) UNIX Programmer's Manual JOVE(1B)

NAME
jove – EMACS style screen editor

SYNOPSIS
jove [–t tagname] [+line] [file1 file2 ...]

DESCRIPTION
JOVE is an interactive display oriented editor which allows one to modify text easily. JOVE stands for Jonathan's Own Version of Emacs. This editor is modeled after the EMACS written at MIT by Richard Stallman. JOVE has tried to stick to the conventions of real EMACS, ...

Figure 5.1. Extract of **man jove** Output.

*****

On the minicomputers, except in the System V environment, **man**'s output is automatically piped through **more** to control scrolling; press the space bar to view the next page. In the System V environment, **man**'s output will scroll down the screen right to its end; to control this scrolling, pipe **man**'s output through **more** (e.g., **man jove | more**) or toggle your terminal's scrolling stop/restart characters (usually **CTRL–S/CTRL–Q**).

On the Crays, **man**'s output is automatically piped through **pg** to control scrolling; press **return** to view the next page.

Both **more** and **pg** provide a repertoire of command line options and interactive commands to control their detailed behavior. Once in **more** or **pg**, interactive commands are available to scroll certain numbers of lines, search for certain strings, and in **pg**, to scroll backwards. Read these commands' man pages for additional information.

Using the **man** command presumes that one knows the name of the command of interest. This is not always the case. Use **man –k** *keyword* or **apropos** *keyword* to determine which commands provide

certain functionality. *keyword* is selected in the usual ways useful for keyword searches. Once a likely command is identified, use **man** *command* to obtain additional information thereon.

On the minicomputers, **man**'s output can be converted to a normal ASCII file by piping it through **col**; for example:

man *command* | **col** –**b** –**f** > *filename*

5.2 Public Information Directory

All the ARLSCF computers have a public directory, **/usr/pub**, which contains information of value to the general user community. The information and files containing it vary from one machine to another, but the following are typical:

manuals	only on the Crays; a list of Cray manuals currently available
notd	notice of the day (current information about the Crays)
notd.bob	month-long log of bob's schedule (down time, test time, etc.)
notd.patton	month-long log of patton's schedule (down time, test time, etc.)
readme.queues	two page summary of the available NQS queues (general information and use)
sys–admin	list of system administrators

One way to obtain this information is to execute:

more /usr/pub/*filename*

5.3 explain

Available only on the Crays, **explain** displays an explanation for many error messages, particularly for errors arising from the use of **cf77**, **cdbx**, and the Fortran and I/O libraries. The syntax is:

explain *groupcode–errornumber,*

where *groupcode–errornumber* is the more cryptic error message provided upon occurrence of the error. Figure 5.2 presents a sample **explain** session. Keyboard entries are emboldened. • indicates a RETURN.

5.4 docview

Available only on the Crays, **docview** provides on-line access to a great deal of documentation. It is more extensive in its coverage of some areas, such as Fortran, than the man pages, but does not yet cover all areas of interest. Inexperienced users should use the **docview** menu to review the list of documents available under **docview**. A particularly useful menu choice is **f** *string*. Execute **man docview** to obtain additional information. Figure 5.3 presents a sample **docview** session. Keyboard entries are emboldened. • indicates a RETURN.

*****

```
prompt> cf77 tryit.f•
 cft77–33 cf77: ERROR
   Unable to read file – tryit.f.
 cft77–25 cf77: 1 CFT77 control statement errors occurred.
 cft77–26 cf77: Compilation aborted.
prompt> explain cft77–33•
```

ERROR: Unable to read file – name.

The operating system was unable to read from the source input file or from the inline file. This usually occurs when there is no file with the specified name or when the file's read permission is not set properly.

For more information, read about compiling in the CF77 Compiling System, Volume 1: Fortran Reference Manual, Cray publication SR–3071.
prompt>

Figure 5.2. Sample **explain** Session.

*****

*****

prompt> **docview•**

D O C V I E W
On-line Documentation System Command Menu

Please enter a command at the menu> prompt.

a[list]	List docnames in alphabetical order
c[list]	List docnames by subject category
d[list]	List docnames by date last submitted
f[ind] [string]	Find keywords and corresponding docnames associated with "string"
p[revious]	Return to the previous command mode
v[iew] [docname][keyword]	View passage "keyword" in document "docname"
w[rite] [docname][keywords]	Write passages specified by "docname" and "keywords"
h[elp] [topic]	Display help for the current screen or a Docview topic or command
m[enu]	Display this menu
q[uit]	Quit from Docview

Enter "help quick" for a quick look at how to use Docview
menu> **q•**
prompt>

Figure 5.3. Sample **docview** Session.

*****

5.5 Electronic Mail

Electronic mail (email) provides a good source of on-line information and help. By sending a mail message to **support**, which is monitored by many people at ARLSCF, the user can obtain information and assistance on questions concerning either the Crays or the minicomputers. It is important that the message specify the computer to which the question or problem pertains. By sending a mail message to **craysupport**, the user can obtain information and assistance on questions concerning the Crays. Responses from either address are almost always obtained within a few days, and usually within a few hours.

5.5.1 Email on ARLSCF Non-Cray Machines

Execute **man msg** and **man send** for detailed information about the mail system. Figure 5.4 illustrates sending a message to **support**. Because the user wishes merely to send a message, he invokes only the **send** subsystem of **msg**. The session includes a request for help and the mail system's response. Keyboard entries are emboldened, and • indicates a RETURN.

*****

prompt> **send**•
SEND (6 Dec 1988 Update #32)
To: **support**•
cc: •
Subject: *user's short descriptive title*•
Type message; end with CTRL–D...

user's message
CTRL–D Command or ?: **?**•
bcc
bye
check spelling
delete body
edit body (using editor)
vedit body (using veditor)
file include
header edit
input more body
program run
quit
review message
send message
set [option] [option value]
Command or ?: **s**•
support@ARL.ARMY.MIL: address ok
Message posted.

prompt>

Figure 5.4. Sending a Message to **support**.

*****

Figure 5.5 illustrates an invocation of **msg**, which is required if anything more than sending is to be done, and which permits **send** subsequently to be invoked.

*****

```
prompt> msg•
MSG (9 May 1989 Update #32)  Type ? for help.
Loading binary box /other/joe/.mailbox . . . . . 37 messages total.
<– ?
Command Summary –– Type only the first letter of a command

MESSAGE HANDLING:                    NAVIGATION:
Answer [message #(s)]                Backup to prev msg & type it
Forward [message #(s)]               Next message & type it
Send (compose) a new message         Headers of [message #(s)]
Type [message #(s)] onto terminal    Go to message #
Delete [message #(s)]                Current message # is typed
Undelete [message #(s)]
Keep [message #(s)]                  /–same as dcn
Y–Resend [message #(s)]
Z–Two window answer

FILE HANDLING:                       MISCELLANEOUS:
Exit and update –– normal exit       Jump into sub-Shell
List [message #(s)] into text file   Xtra user options
Overwrite old file                   : current date and time
Put [message #(s)] into msg file     ; ignore rest of line
Quit –– fast exit                    @ Undigestify
Read another msg file                ! sub-Shell
Move [message #(s)] into msg file & mark as deleted

Examples of message #(s) are:  42  1–4  2,5,7–.  1–5,7–@  c  32–$
<–
```

Figure 5.5. An Invocation of **msg**.

*****

5.5.2 Email on patton and bob

There are two distinct mail systems on the ARLSCF Crays. Neither will accept incoming mail from any other machine. The Cray Research Inc. **mail** permits mail to be sent to accounts on the originating Cray and to accounts on other machines which are not ARLSCF Crays. The UNIX **msg** and **send** are useful only to send mail from the Cray to accounts on other machines which are not ARLSCF Crays. The Cray–2 can communicate only with machines on its classified net.

Cray Research Inc. **mail** can send email to accounts on the originating Cray and to accounts on other machines. For local mail on a Cray, a user name suffices as an address. For outgoing mail, the address must be of form *user@nodename*. For example, to send from the Cray X–MP to user **joe** who receives mail on one of the ARLSCF minicomputers (keyboard entries are emboldened, and • indicates a RETURN):

patton> **mail joe@arl.army.mil•**
user's message
CTRL–Dpatton>

To send a preexisting file:

```
patton> mail joe@arl.army.mil<filename•
patton>
```

The UNIX-style **send** can be used only to send mail from the Cray to other machines which are not ARLSCF Crays. No **send** mail is automatically dispatched from the Cray; any user must execute

/usr/mmdf/deliver –w –cbrlnet,smtp

to cause all **send** mail accumulated to that point to be dispatched. The **msg** and **send** manual pages are not available on the Crays.

5.5.3 Email and File Transfers

In general, email should not be used for file transfers. This is especially important for large files, and for transfers involving a Cray, because such activity can overload a machine's **/usr** file system. The NQS (see the chapter, "Batch Jobs"), invoked by the **qsub** command, provides a number of options and defaults to send information to the originating or other user at certain points during processing of the submitted job. At the ARLSCF, these transfers are accomplished through the email system; therefore, the user must not permit large amounts of information to be so transmitted. **rcp** and **ftp** (see the chapter, "File Transfers") are the proper and efficient commands to effect file transfers.

6. File Storage – On- and Off-Line

At the ARLSCF, disk space is limited, but new files are always being created. Therefore, some mechanism for "recycling" disk space is necessary. In addition, a mechanism to maintain backup copies of files is necessary in order to recover from losses due to system problems and inadvertent destruction by users. With the exception of file migration and **t3480** archiving, the following mechanisms are available on all ARLSCF computers; file migration and **t3480** archiving are available only on the Crays.

- file compression
- file migration
- **/usr/tmp** and **/tmp** directories
- user action
- users' own tape archives
- system backups

Details vary from one machine to another and are best obtained from the **man** pages and by discussion with a particular machine's system administrator, using email. Detailed information herein applies specifically to the Crays.

6.1 File Compression and Decompression

compress *filename* produces file *filename*.**Z** which, typically, is only 40% to 60% as large as the original. **uncompress** *filename*.**Z** restores a compressed file to its original form. **compress** requires that sufficient space be available for the compressed and uncompressed files to coexist. Upon successful completion, the original file is removed. Without filenames, both commands can appear in pipes, eliminating the need for both forms of the file to coexist. These commands are recommended over **pack** *filename* and **unpack** *filename*.**z** (note the lower case **z**) because they execute more quickly, achieve greater compression, and can be used in pipes.

6.2 File Migration

The UNICOS data migration facility attempts to maintain availability of disk space on the ARLSCF Cray computers by moving selected files to off-line media (at the ARLSCF, tape). Files which are migrated are still catalogued in their original directories, but the files themselves are no longer on-line.

6.2.1 Operation of System Initiated Migration

When a file system's free space falls below a critical level, file migration is initiated. Candidate files are those older than 1 day, larger than one block (4096 bytes, 512 words), and not exempted from migration (see **.keep** file in the following section, "User Interface..."). Within those constraints, files will be copied to tape (two copies for reliability) and deleted from disk, largest and oldest files first, until a certain amount of disk free space is restored.

6.2.2 Pros of File Migration

File migration frees disk space, allowing computer operations to continue.

File migration is completely transparent, except that a user

- can determine whether a file has been migrated (**ls –l** displays **m** rather than – in the leftmost column when a file is migrated).
- can explicitly force particular files to be migrated and recalled.
- may experience delays in program execution/interactive response.

6.2.3 Cons of File Migration

The system can be costly to operate, in terms of reduced system responsiveness and use of assets (tape drives) when a migration is being done, when migrated files are being recalled, and when the migration database and tape library are maintained and consolidated.

Consider the effect of introducing into a user file system (e.g., **/other**, **/amsaa**) a file sufficiently large that migration is triggered. Files are migrated as indicated previously. The migration itself increases overhead and ties up tape drives. Of the files which migrate, users will recall those which are needed, further increasing system overhead and tying up additional tape drives. The larger the migration library and databases, the worse will be the increases in overhead. Furthermore, it is the larger files which are migrated; when a sufficient number of them are recalled, migration will again be triggered. It has happened (infrequently) in the past that a particularly vicious cycle is entered, wherein little but migration and recall occur. Some solutions to these problems are discussed in the following paragraphs.

Each file in a file system requires an inode, wherein is kept information permitting the operating system to find that file on demand. There are only a certain number of inodes; thus a large number of small files could "fill" a file system, even though adequate disk space remain. File migration is not triggered by running out of inodes, nor does its operation reduce the number of inodes in use because migrated files are still catalogued in their original directories.

6.2.4 Current Status of the File Migration System

To ensure reliability, two copies are kept of every file migrated.

Currently, the ARLSCF migrated file library consists of approximately half a terabyte of information, constituting approximately a third of a million files stored on approximately 3000 tapes. The number of files held in migration and the number of tapes used to hold those files continue to grow. The larger the number of migration tapes, and the greater the number of files in the migration database, the less efficiently the migration software operates. In addition, there is a limit the the number of migration tapes which can be maintained, due simply to space available.

Periodically, a reconciliation of files known to the migration database against files on-line and on tape is required. Periodically, the entire pool of migration tapes must be merged to eliminate space wasted by partially filled tapes, files deleted by users but still on tape, and multiple copies of files. These actions consume significant amounts of computer time and other assets.

6.2.5 User Interface with the File Migration Facility

To determine if a file is migrated, execute **ls –l**. A migrated file will display an **m** in the leftmost position of the output.

To exempt certain files from migration, create in your home directory a file named **.keep**, and in that file place, left-adjusted, one per line, the full path name of each file to be exempted. The files named in **.keep**,

starting with the first, and continuing up to but not including the file which exceeds a certain limit, are exempted from system initiated migration. Currently, the limit is 2000 blocks, a block being 4096 bytes or 512 words.

To cause specific files to be migrated, execute **dmput** *filenames*. **Do not use migration as an archival system**. A legitimate and beneficial use of **dmput** is to cause migration of files which are too new to be migrated automatically and which are large enough to be significant contributors to the need for migration. When this is not done, such files may cause the migration of many other files. In turn, their migration, subsequent recalls, and contribution to resource consumption during migration library and database maintenance are the prime contributor to poor system response and large overhead. In a worst case, there can be so many requests for files from the migration library, and additional migrations generated as their restorations consume file system space, that the system essentially comes to a standstill. See also the following section, "**/usr/tmp** and **/tmp** Directories."

To recall a migrated file, execute **dmget** *filenames* (execute **man dmget** for additional information). This is an explicit recall and can conveniently be executed in background to avoid waiting while the files are recalled.

Migrated files can be recalled implicitly by attempting to access them (e.g., **cat** *filename*). Implicit recalls ultimately generate equivalents of **dmget**s, one for each implicit recall. If more than one file is to be recalled, it is much more efficient to explicitly use one **dmget** with the entire list of files. The **dmmode** command modifies the system's response to an implicit recall. **dmmode 0** sets a user's environment so that an implicit recall will be rejected and the message *filename*: **File off-line, no automatic retrieval** issued. **dmmode 1** sets a user's environment so that an implicit recall is honored and a delay ensues while the file is recalled. The **dmmode** command typically is placed in a user's **.profile**, **.cshrc**, or **.tcshrc** file. The system default is **dmmode 1**.

When a file is recalled, explicitly or implicitly, in foreground, there is a delay while the database is searched, the tape obtained and mounted, and the file read in. These delays can be quite short, but are very strongly influenced by the level of activity of the migration system (especially when there is contention for resources) and by the availability of operators.

6.3 /usr/tmp and /tmp Directories

Unlike "normal" user directories, **/usr/tmp** and **/tmp** are available to all users and are not subject to file migration. As indicated by their names, they are intended to hold files temporarily — 4 days in **/usr/tmp** on the Cray X–MP, 14 days in **/usr/tmp** on the Cray–2, and until the next reboot in **/tmp**, unless the directories fill up, in which case files must be removed. Every effort is made to contact owners of those files before they are removed.

The advantage of using these directories is that large files placed therein do not trigger migration. Files can be archived from these directories.

6.4 User Action

User directories can reach a critically low free space condition, triggering migration, which reduces system responsiveness. In addition, migration occurs frequently enough that the migration library and its database are already almost twice as large as ARLSCF staff had ever imagined they could become. They continue to grow. This causes difficulty for everyone, especially users, who occasionally suffer very long waits to recall files. In addition, library and database maintenance, while infrequent, consume resources: library tape merges degrade responsiveness, and reconciliation of the database and library with each other and with on-line file systems requires dedicated time.

When a user operates in such a way that his own files are not migrated, he benefits himself in that his computer activity is never delayed by (occasionally long) waits for the recall of his own migrated files. In

addition, he benefits the entire system because there is no increase in overhead due to the migration and recall of his own files. Finally, he benefits the entire system because the migration database and library remain smaller than would be the case, were his own files migrated.

The migration system is not, and does not operate well as, an archival system. By far the best, and a highly efficient, solution to the problems associated with data migration is for users to realize that their files consume scarce and valuable assets, even when they are migrated, and to act accordingly:

- Remove files (**rm**, **rm –r**), including migrated files, which are no longer needed. Removal of migrated files will be especially beneficial to the migration facility, which contains many files not accessed for very long times, because it reduces the sizes of the database and of the library.
- Archive on-line files (see the following section, "Archives") which are infrequently used, but of continuing value, and then remove (**rm**) them from disk.
- Archive large on-line files (see the following section, "Archives") even if they are frequently used, and then remove (**rm**) them from disk.
- Archive already migrated files, and then remove (**rm**) them from disk. (see the following section, "Archives"). Do not perform this action without first contacting the ARLSCF support staff because the attempt to be a "good computer citizen" may be counterproductive. Every migrated file being archived will first be recalled, thus reducing responsiveness and consuming disk space which may trigger other file migrations. Special techniques avoiding these problems are available to the support staff.
- Create, recall from archive, and recall from migration[†] large files[††] into **/usr/tmp** and **/tmp** because these directories are not subject to migration.

6.5 Archives

There is no system archiving facility at the ARLSCF. There are, however, several utilities available for users to create, use, and maintain their own personal archives on tape. The user may choose to keep the tapes physically in his possession, or the ARLSCF staff will keep the tapes in environmentally controlled areas convenient to the appropriate computers. Cray–2 tapes, all of which are presumed to be classified until certified otherwise, may not be removed from that site without prior coordination with ARLSCF staff.

Because users' personal archives invariably are small (at least in comparison to the migration library), there is no need for elaborate database-driven schemes to access files, nor for elaborate tape merging and reconciliation schemes to maintain the archives. In addition, users are able to commit to tape and remove from disk their own files according to schemes which are more rational and usage driven than all-purpose schemes like the one used by file migration.

The utilities used for archiving are available to users to copy files to tape for transmittal to other locations, and also to read tapes received from other locations. ARLSCF staff members are available for assistance. Because there is such a wide variety of logical and physical tape formats, it is good practice to discuss intermachine tape transfers with ARLSCF staff.

---

† Users can recall migrated files only into the directories from which they were migrated. Depending on file system status, however, even the momentary introduction of a large file into a file system can trigger migration. ARLSCF support staff can recall migrated files into other than their original directories, and will be pleased to assist in the recall of large files from migration into **/tmp** or **/usr/tmp**.

†† How large is "large?" File system migration is triggered when free space on a file system falls below 30% of its total size. Therefore, users dealing with large files ought to compare file sizes to file-system-free-space minus .3*file-system-total-size, to determine if recalling a particular file will definitely or probably trigger migration. The **ls –l** command displays file size in bytes. The **df –t** command displays file system total and free space in blocks, a block being 4096 bytes.

6.5.1 t3480

Available only on the Crays, and ultimately invoking **cpio**, **t3480** is a UNICOS shell script which reads/writes files to/from multiple volumes of IBM 3480 cartridge tape(s). Each cartridge holds approximately 200 megabytes of information. These cartridges are recommended for archiving Cray files and for file transfer between the two Crays, but not for transmittal of files to other locations. Usually, these cartridges are not released into users' possession.

The user maintains his own audit of his archive. To assist in this maintenance, **t3480** appends a descriptive log of all its tape writes (except when **–T** is specified) in the file **.tapelog** in the user's home directory. Do not let this file grow too large (or else, enter it into **.keep**; see the preceding section, "File Migration"), lest it be migrated and thus delay **t3480**. When writing, **t3480** first writes the tape, then rewinds and reads it, making entries in **.tapelog**. Absence of expected entries in **.tapelog** indicates some failure in the writing process. **t3480** writes or overwrites, but does not append, tapes.

Usage of **t3480**:

t3480 –r|–w|–n|–t [–T] –v vol [*files*]

–r read *files* from the cartridges and write them to disk, creating directories and subdirectories as needed, overwriting files with the same path names as files read.

–w writes *files* to previously written cartridges, which already have physical and logical labels. Overwrites all existing data.

–n writes *files* to new cartridges, which have neither physical nor logical labels.

–t writes to standard output a list of the files on the cartridges.

–T modifies the action of –r|–t to read or catalogue a logically unlabeled tape, and of –w|–n to write or overwrite a physically and logically unlabeled tape (a new tape). Only one cartridge is processed, so writes cannot be for more than 200 megabytes. Useful with tapes from/to another machine. Because of security considerations, creation of tapes intended for transfer from ARLSCF's Cray–2 requires prior coordination with ARLSCF staff.

–v *vol* *vol* is the user-supplied portion of the 6-character tape label: 3 letters (**t3480** converts lower case to upper case) and digits. **t3480** provides 1 character identifying the host machine, 2 two digits specifying the cartridge count.

files If no files are listed, all files and subdirectories in the current directory are written to or read from the tape. **t3480** expands UNICOS wild card characters such as * and **?**.

Examples:

- Write files **abc** and **def** on new cartridges on patton.arl.army.mil:

  t3480 –n –v 123 abc def

  The resulting label is **W12301**. Both files fit on one tape; **W** indicates a cartridge made on patton.

- Overwrite all files in directory **work** and its subdirectories on previously used tapes labeled **WZBC01**, **WZBC02**, ... (unneeded tapes become unreadable):

  cd work
  t3480 –w –v ZBC

- Restore the directory **work**, but as directory **play**;

  cd $HOME
  mkdir play
  cd play
  t3480 –r –v ZBC

6.5.2 tar and cpio

tar and **cpio** can be used to save/restore (multiple) files and directories to/from tape. They are particularly useful for saving multiple files and directories on the same tape, **tar** providing an option-selectable append capability. These facilities are of value to users whose needs are not met by **t3480**, particularly when tapes are created for transfer to other machines (except that **t3480** is preferred for transfers between the ARLSCF Crays), and to users who choose to deal with tapes on machines other than the Crays, even when they are dealing with Cray files. For detailed information, execute **man tar** or **man cpio**.

6.6 Tape Usage

The preferred method for reading/writing tapes on the ARLSCF Crays, for the maintenance of archives, and for transfer between the Crays, is **t3480**.

For other purposes, tapes containing Cray files are found by some users to be more conveniently read/written on other ARLSCF machines (minicomputers, workstations, etc.), or on machines at other sites. Typically, these users make use of **rcp** or **ftp** to transfer the files of interest to/from the "other" machine, and **tar** or **cpio** to read/write the tape. Such methods require the user to consider how the tapes physically will be mounted on the desired machine.

The remainder of this section is intended for users who choose to read/write tapes on the ARLSCF Crays by means other than **t3480**. The Cray Tape Controller requires that a setup procedure be followed whenever tapes are used on the Crays (this procedure is built into **t3480**).

The ARLSCF provides two types of tape drive, the IBM 3420 (round) and the IBM 3480 (cartridge). Before a drive can be accessed, **rsv TAPE** or **rsv CART** must be executed to reserve a tape resource, round or cartridge, respectively. A user must not execute **rsv** when he has already reserved resources; **rls –a** releases all of a user's resources. If the requested resource is not available, or if resources are already reserved, an error message is issued. Once the **rsv** has been successful, the user executes a **tpmnt**, which requests the operator mount a tape on a drive, provides additional information to the tape subsystem, and creates a user-named file in **/tmp**. Finally, the user executes a command to read/write from/to that **/tmp** file which acts as an image of the tape. It is important that the user execute **rls –a** upon completion, else the resource remains unavailable to other users. For detailed information on **rcp**, **ftp**, **tar**, **cpio**, **rsv**, **rls**, and **tpmnt**, execute the **man** *command*.

For example, to display a 1600 bpi (bits per inch) round tape's contents on the monitor:

```
rls –a
rsv TAPE
tpmnt –g TAPE –d 1600 –p /tmp/scratch –v VOL1 –l al
cat /tmp/scratch
rls –a
```

For example, to use **tar** to write the entirety of directory **abc** to a cartridge tape:

```
rls –a
rsv CART
tpmnt –g CART –b 4096 –p /tmp/cart –v VOL1 –l al
tar cvbf 8 /tmp/cart abc
rls –a
```

Some other commands of interest when using tapes are:

tpstat	display the status of all tape devices.
msgi	send an informative message to the operator.
msgr	send an informative message to the operator and require acknowledgement.

For additional information, see the Cray publication, *UNICOS Tape Subsystem User's Guide*, SG–2051 6.0.

6.7 Backups

All publicly accessible machines at the ARLSCF are incrementally backed up to tape. One copy is made, during low-use hours at night. In general, all user directories are backed up. In general, a level 0 backup (all files) is done monthly; level 1 (all files with mod dates later than the level 0), weekly; and level 2 (all files with mod dates later than the level 1), daily.

On the Cray X–MP, backups are done on the first Sunday, every other Sunday, and every other day, respectively. Level 0 tapes are retained for 3 months, and levels 1 and 2, for 6 weeks. In addition, **/usr/tmp** is level 0 backed up daily, and the tapes retained for 4 days.

On the Cray–2, backups are done on the first Thursday, every other Thursday, and every other day, respectively. Level 0 tapes are retained for 3 months, and levels 1 and 2, for 6 weeks. In addition, **/usr/tmp** is level 0 backed up infrequently, on an *ad hoc* basis and by special arrangement.

In general, on the minicomputers, backup tapes are retained for 3 months (level 0) and 4 weeks (levels 1 and 2). For specific details on a particular minicomputer, contact its system administrator by email.

Intentionally Left Blank

7. File Transfers

This chapter presents several methods for transferring files among computers at the ARLSCF, or between other sites and the ARLSCF. The transfers are electronic over a network or manual by physically transporting tapes. More complete discussions of the techniques can be found in the on-line manual pages (discussed in the "On-Line Information" chapter of this document) on ARLSCF computers.

7.1 Electronic Transfers

Electronic file transfers can be effected by using **ftp**, **rcp**, or **kermit**. These utilities can be used among machines at the ARLSCF, and between the ARLSCF and other sites.

Electronic mail may be used to transfer files, but it is inefficient for this purpose, and large files will severely burden the mail system.

7.1.1 ftp

ftp (file transfer protocol) permits file transfers among diverse machines running diverse operating systems. It is interactive and has on-line help. Because of its extensive command repertoire, it provides considerable flexibility in the transfer of files. Many sites (but not ARLSCF) provide a "guest" account (login name: **anonymous**) so that users with no account on the remote machine can still copy files therefrom.

Figure 7.1 presents an example wherein **ftp** is used to transfer file **joefile** from **adm.arl.army.mil** to **patton.arl.army.mil**. User **joe** is executing **ftp** on adm and has an account on patton. Keyboard entries are emboldened. • indicates a RETURN.

*****

prompt> **ftp patton.arl.army.mil•**
Connected to patton.arl.army.mil.
220 patton FTP server (Version 5.2 Fri Sep 7 14:09:58 CDT 1990) ready.
Name (patton.arl.army.mil:joe): **joe•**
331 Password required for joe.
Password:*user enters password, not echoed to screen•*
230 User joe logged in.
ftp> **put joefile•**
200 PORT command successful.
150 Opening ASCII mode data connection for joefile.
226 Transfer complete.
local: joefile remote: joefile
532 bytes sent in 0.048 seconds (11 Kbytes/s)
ftp> **bye•**
221 Goodbye.
prompt>

Figure 7.1. **ftp** File Transfer.

*****

For a transfer in the opposite direction (patton to adm, user still executing **ftp** on adm), the only difference in the user's actions would be to enter **get** rather than **put**. In certain cases, the user can abbreviate the remote machine name. In the example, **patton.arl.army.mil** could have been abbreviated to **patton**.

Figure 7.2 shows a list of **ftp** commands obtained from within **ftp** by typing **help** or **?**. • indicates a RETURN. Descriptions of these commands may be obtained by executing **man ftp**.

*****

prompt> **ftp•**
ftp> **?•**
Commands may be abbreviated. Commands are:

!	cr	macdef	proxy	send
$	delete	mdelete	sendport	status
account	debug	mdir	put	struct
append	dir	mget	pwd	sunique
ascii	disconnect	mkdir	quit	tenex
bell	form	mls	quote	trace
binary	get	mode	recv	type
bye	glob	mput	remotehelp	user
case	hash	nmap	rename	verbose
cd	help	ntrans	reset	?
cdup	lcd	open	rmdir	
close	ls	prompt	runique	

ftp> **bye•**
prompt>

Figure 7.2. **ftp** Help Screen.

*****

7.1.2 rcp

rcp (remote copy utility) is the most convenient way to transfer files among UNIX machines, but it is not nearly so flexible as **ftp**. There must be an appropriate **.rhosts** file in the user's home directory on the remote machine. A **.rhosts** file contains a list of the names of machines from which the remote machine might be addressed, one name per line, each beginning in column 1. For example, to perform as in the **ftp** example, but using **rcp**, user **joe** would have a **.rhosts** file in his home directory on patton, containing at least the line

adm.arl.army.mil

Entering

rcp joefile patton:joefile

has exactly the same effect as the **ftp** example, and the circumstances of its execution are exactly as in that example, with the addition that user joe has an appropriate **.rhosts** file in his home directory on patton.

Were the desired transfer to be in the opposite direction (patton to adm, user still executing **rcp** on adm), the user would enter

rcp patton:joefile joefile

In short, the original file is always the first argument, and the copy of the file, the second. Whichever is on the remote machine is prefixed with the name of that machine and a colon. Either file name may be

replaced with a full path name or a relative path name – relative to the current directory on the local machine, or to the user's home directory on the remote machine. The copy file name may be replaced with an existing directory name, in which event the copy of the file is placed in that directory with its original name. The **–r** option specifies recursive use; that is, the original name specifies a subtree rooted at that name, and the copy name specifies a directory which is created and into which is copied the subtree.

7.1.3 kermit

kermit is useful not only to transfer files, but also to enable the local computer to emulate a terminal on the remote machine. The local and remote hosts may be of diverse types running diverse operating systems. In this role, **kermit** is particularly useful when the local machine is a PC, Macintosh, etc., and communicates over telephone lines using a modem. Establishing communications requires that the local **kermit** be "set up" with parameters dependent upon the hosts and modems involved. Once **kermit** communications are established, the local machine emulates a terminal connected to the remote host, whose repertoire of commands can be executed. In particular, **kermit** can be invoked on the remote host and commanded to send/receive files. An escape is then made to the local **kermit**, which is commanded to receive/send files.

7.1.4 Electronic mail

In general, electronic mail (email) should not be used for file transfers. This is especially important for "large" files, and for transfers involving a Cray, because such activity can overburden a machine's **/usr** file system.

The NQS (see the chapter, "Batch Jobs"), invoked by the **qsub** command, provides a number of options and defaults to send information to the originating or some other user at certain points during processing of the submitted job. At the ARLSCF, all these tranfers are accomplished through the email system; therefore, the user must not permit large amounts of information to be so transmitted.

With awareness of the potential for problems, users can use email occasionally to transfer small files (less than 1 megabyte) at the ARLSCF or between the ARLSCF and remote sites. Transfers of this sort are only indirectly to a particular machine; rather, they are to a particular addressee and, in turn, to whichever machine is designated as the addressee's mail receiver. This provides the convenience that the sender need not consider whether he has an account on the target machine, nor whether he has the permissions necessary to write the file at its destination, nor even what the destination ought to be.

7.1.4.1 Email Transfers Not Involving the ARLSCF Crays

Execute **man msg** and **man send** for additional information about the email system. Figure 7.3 illustrates sending a file in the current directory to user **joe**; were the file in some other directory, a full or relative path name would be needed. Because the sender wishes merely to send a message, he invokes only the **send** subsystem of **msg**. The session includes a request for help and the mail system's response. Keyboard entries are emboldened, and • indicates a RETURN.

*****

prompt> **send**•
SEND (6 Dec 1988 Update #32)
To: **joe**•
cc: •
Subject: **the file you wanted**
Type message; end with CTRL–D...

CTRL–D Command or ?: **?**•
bcc
bye
check spelling
delete body
edit body (using editor)
vedit body (using veditor)
file include
header edit
input more body
program run
quit
review message
send message
set [option] [option value]
Command or ?: **f**•
File: *filename*•
...included
Command or ?: **s**•
joe@ARL.ARMY.MIL: address ok
Message posted.

prompt>

Figure 7.3. sending a Message.

*****

7.1.4.2 Email Transfers Involving the ARLSCF Crays

There are two distinct mail systems on the ARLSCF Crays. Neither will accept incoming mail from any other machine. The Cray Research Inc. **mail** permits mail to be sent to accounts on the originating Cray and to accounts on other machines which are not ARLSCF Crays. The UNIX **msg** and **send** are useful only to send mail from the Cray to accounts on other machines which are not ARLSCF Crays. The Cray–2 can communicate only with machines on its classified net.

Cray Research Inc. **mail** can send email to accounts on the originating Cray and to accounts on other machines. For local mail on a Cray, a user name suffices as an address. For outgoing mail, the address must be of form *user@nodename*. For example, to send from the Cray X–MP to user **joe** who receives mail on one of the ARLSCF minicomputers (keyboard entries are emboldened, and • indicates a RETURN):

patton> **mail joe@arl.army.mil**•
user's message
CTRL–Dpatton>

To send a preexisting file:

patton> **mail joe@arl.army.mil**<*filename*•
patton>

As an aside, note that assistance can be requested of **craysupport** with the address **craysupport@arl.army.mil**.

The UNIX style **send** can be used only to send mail from the Cray to other machines which are not ARLSCF Crays. No **send** mail is automatically dispatched from the Cray; any user must execute

/usr/mmdf/deliver –w –cbrlnet,smtp

to cause all **send** mail accumulated to that point to be dispatched. The **msg** and **send** manual pages are not available on the Crays.

7.2 Manual Transfers

In general, files can be transferred between machines and sites on 9-track tapes. At the ARLSCF, such tapes can be read and written with the **dd**, **cpio**, **tar**, and **ansir** (read only) utilities. We strongly recommend that users who are not themselves expert in tape operations, and who wish to transfer files between sites via tape consult with both the writing and the reading site support staffs before the tape is written. When a 9-track tape written elsewhere with utilities other than **tar** is to be read at the ARLSCF, the requestor must be prepared to specify the details of the tape structure (e.g., block size, record length, label information, mapping scheme).

Files can be transferred between certain machines and certain sites on IBM 3480 cartridge tapes. At the ARLSCF, such tapes can be read and written with the **t3480** utility, which invokes **cpio** with appropriate options and issues other commands.

7.2.1 t3480

Available only on the Crays, and ultimately invoking **cpio**, **t3480** is a UNICOS shell script which reads/writes files to/from multiple volumes of IBM 3480 cartridge tape(s). Each cartridge holds approximately 200 megabytes. These cartridges are recommended for archiving Cray files and for file transfer between the two Crays, but not for transmittal of files to other locations. Usually, these cartridges are not released into users' possession. Except when **–T** is used, **t3480** will write across tape boundaries as necessary to produce multivolume tape files. Detailed **t3480** information is available in the man pages.

For example, the command

t3480 –n –v *vsn* **–T** *filenames*

specifies a new cartridge (which implies a write operation), with neither physical nor logical label (**–n**), specifies the user portion of the tape label/volume name to be *vsn* (**–v**), and specifies that the purpose of the tape is to transfer files between the two ARLSCF Crays (**–T**). In addition, the **–T** option suppresses creation and update of the user's **.tapelog** file and restricts the writing to a single cartridge. Replacing **–n** with **–r** will read the resulting cartridge.

Reading and writing tapes always involves coordination with the machine operators or other support staff members, if only to request that tapes be mounted (**t3480** automatically issues such requests). When, however, tapes are written to transfer files from the ARLSCF Cray–2, security considerations mandate that the user communicate directly with support staff.

7.2.2 dd

The **dd** (data dump) program is the preferred utility for reading and writing tapes transferred between sites. **dd** can read and write tapes in many different formats for compatibility with many different computers and operating systems. More complete information is available in the man pages.

For example, on a minicomputer, the command

dd if=*filename* **of=/dev/rmt0 obs=1320 cbs=132 conv=block**

writes file *filename* onto the tape on drive **0** with a block size of 1320 bytes (characters) and fixed length records of 132 characters. To read that tape into file *filename* requires the exchange of the values associated with the **if** and **of** options, the replacement of the option name **obs** with **ibs**, and change of the **conv** option value from **block** to **unblock**.

dd on the Crays does not support **block/unblock** conversion. If it is necessary to read/write **dd** tapes using this feature with Cray files, we recommend those files be transferred to an appropriate minicomputer and the tape operations be performed there. Another option is to obtain a program which can read and write under under format control to convert between fixed length and variable length records, and use it in conjunction with **dd** on the Crays. For example, on patton, the commands

rsv TAPE
tpmnt –l nl –g TAPE –p *tmp.tape.link* **–v** *eid* **–b 1320**
pad < *filename* **| dd of=***tmp.tape.link* **bs=1320**
rls –a

write file *filename* onto an unlabelled tape with external identification *eid*. The tape is written with a block size of 1320 bytes (characters). The result is the same as on the minicomputer, given that program **pad** reads file *filename* one variable length record at a time, pads that record to 132 characters with blanks, and then writes it to standard output. To read that tape into file *filename* requires the **dd** command line be replaced with

dd if=*tmp.tape.link* **bs=1320 | unpad >** *filename*

where **unpad** strips the trailing blanks from the fixed length records.

7.2.3 cpio

cpio is useful when the writing and the reading are both done on UNIX systems. **cpio –o** reads from standard input to obtain file names and writes those files to standard output. **cpio –i** reads from standard input information previously processed by **cpio –o**, extracts the individual files specified as arguments (with shell expansion of metacharacters, default *), and places those files into the current directory. More complete information is available in the man pages. We recommend that users not expert in tape operations avoid **cpio** (except as invoked through **t3480**).

7.2.4 tar

tar (tape archive) is useful when the writing and the reading are both done on UNIX systems. Its use is particularly convenient when an entire directory or subdirectory structure is to be written to or copied from tape. More complete information is available in the man pages.

For example, on a minicomputer, the command

tar –c .

copies the entire subtree rooted at the current directory to the tape mounted on the default drive and

tar –x .

restores that subtree, rooting it at the current directory. Ownership, group, permissions, and modification times can be preserved or changed. Suffixing **v** to the option in either command makes the command verbose.

On the Crays, **tar** operates similarly, but the **tar** command line must be augmented with other commands, as is the case for **dd** (discussed previously).

7.2.5 ansir

ansir (ANSI read), not available on the Crays, reads ANSI labeled tapes. By default, it opens in interactive mode, wherein the user is questioned about various characteristics of the tape. The user must know the character set, record length, block size, and number of files on the tape.

Figure 7.4 depicts extraction of two files from a previously written tape, user entries being emboldened and • representing a RETURN:

*****

```
prompt> ansir /dev/rmt0

*** At file number 0:
VOL1–label missing

*** At file number 1:
HDR1–label missing
HDR2–label missing
Tape mark missing

      I shall have to ask some questions; to get more information,
      answer a question with a single question mark (?).

Please type your file name: prog23
Please type your character code: ASCII
Default record format: U
Please type a new record format, or press RETURN: •
Default block length: 32766
Please type a new block length, or press RETURN: •
Default buffer offset: 0
Please type a new buffer offset, or press RETURN: •
EOF1–label missing
EOF2–label missing
Tape mark missing

*** At file number 2:
HDR1–label missing
HDR2–label missing
Tape mark missing
Please type your file name: data23
Default character code: ASCII
Please type a new character code, or press RETURN: •
Default record format: U
Please type a new record format, or press RETURN: •
Default block length: 32766
Please type a new block length, or press RETURN: •
Default buffer offset: 0
Please type a new buffer offset, or press RETURN: •
EOF1–label missing
EOF2–label missing
Tape mark missing

*** At file number 3:
HDR1–label missing
HDR2–label missing
Tape mark missing
Please type your file name: CTRL–C
prompt>
```

Figure 7.4. Extracting Files from Tape.

*****

8. Batch Jobs

This chapter presents an overview of the various methods for submitting batch jobs at the ARLSCF. The greater portion of the chapter is devoted to the NQS, used to submit a batch job to one of the Crays from that Cray. Other methods, such as interactive batch processing by executing in background, also are discussed briefly.

8.1 NQS

NQS is available only on the Crays. It permits submission of jobs exceeding the resource limits imposed on interactive jobs, both synchronous and asynchronous. NQS jobs are automatically checkpointed before intentional system shutdowns, thus allowing recovery when the machine returns. A detailed discussion of NQS can be found in the on-line manual pages and in the Cray publications, *UNICOS Primer*, SG–2010 6.0, and *UNICOS User Commands Reference Manual*, SR–2011 6.0.

8.1.1 The Queues

The **qsub** command is used to submit jobs into one of four queues:

express	Intended for high priority jobs. Charged double the normal rate.
deferred	Intended for low priority jobs. Charged half the normal rate. Receives little execution time unless the system is idle.
debug	Intended for short, quick turnaround jobs typical of debugging. Charged the normal rate.
crayque	The default. Intended to contain the great bulk of jobs submitted. Charged the normal rate. Jobs are automatically placed in subqueues based on the time and memory requirements specified in the job file. (Users can determine the memory required for a job by testing the executable with the **size** command. See the **size** man page for detailed information.)

Rates are specified in Appendix A.

In general terms, the queue structures on both the Cray–2 and the Cray X–MP are the same, but their specific characteristics reflect the hardware and usage differences of the two machines.

A job's **nice** value influences its execution priority; the higher the value, the **nicer** it is to other jobs and, hence, the longer it takes to complete, given that there is contention for system resources.

8.1.1.1 Cray–2 Queues

queue	permanent file space (MW) per process	permanent file space (MW) per request	memory size (MW) per process	memory size (MW) per request	nice both	CPU time (sec) both
express	512	512	224	unlim	1	unlim
deferred	unlim	unlim	224	unlim	19	unlim
debug	512	512	224	unlim	4	180
crayque						
xsmall	128	256	4	4	6	3600
small	128	256	8	8	6	5400
medium	128	256	64	64	8	10800
large	256	512	128	256	10	72000
xlarge	unlim	unlim	224	unlim	10	172800
huge	unlim	unlim	224	unlim	10	unlim

8.1.1.2 Cray X–MP Queues

The express queue, as well as crayque, is split into subqueues. In addition, there may be special queues, not available to the general user community, created for extraordinary circumstances. The deferred queue and all 4 MW (actually, 3.6 MW) queues except debug execute only between 4 PM and 8 AM.

queue	permanent file space (MW) per process	permanent file space (MW) per request	memory size (MW) per process	memory size (MW) per request	nice both	CPU time (sec) both
express						
express_2m	unlim	unlim	2.0	2.0	1	unlim
express_4m	unlim	unlim	3.6	3.6	1	unlim
deferred	unlim	unlim	3.6	3.6	19	unlim
debug	512	512	3.6	3.6	6	180
crayque						
q2m_10m	128	256	2.0	2.0	6	600
q2m_1h	128	256	2.0	2.0	8	3600
q2m_2h	128	256	2.0	2.0	8	7200
q2m_20h	128	256	2.0	2.0	8	72000
q2m_unl	128	256	2.0	2.0	8	unlim
q4m_3h	512	512	3.6	3.6	10	10800
q4m_20h	unlim	unlim	3.6	3.6	10	72000
q4m_unl	unlim	unlim	3.6	3.6	10	unlim

8.1.2 Submitting NQS Jobs

Users must specify memory and time requirements with options on the command line, or with the same options on # **QSUB** lines at the beginning of the job file. The system uses these values for three purposes:

- To verify that a job may enter the specified queue.
- To place a job in the appropriate subqueue, if any. The subqueue selected is the first one encountered (in the order of the preceding tables) whose per-process and per-request limits are not exceeded by job requirements.
- To place limits on the resources a job may consume during execution. Users should not merely "guess high" to avoid being terminated during execution because so doing may result in long turnaround times.

8.1.2.1 Options – Security

The **–u** option permits a password to be supplied as part of the job. Do not use this option to provide a password. Users are directed not to include any password on the **qsub** command line or in the job file. Under no circumstances should any user ever enter any password into any file. The proper mechanism for validation of an NQS submission is through a **.rhosts** file.

8.1.2.2 Options

There are approximately 30 options to the **qsub** command. A brief introduction follows; detailed information may be found in the references already cited.

The following displays an NQS job file, and the command used to submit it on the Cray–2:

```
bob> cat subfile
# QSUB-q crayque        # submit file to crayque
# QSUB-eo               # standard error to standard out
# QSUB-lm 8mw           # per-process memory limit
# QSUB-lM 8mw           # per-request memory limit
# QSUB-lt 7500          # per-process CPU time limit
# QSUB-lT 7500          # per-request CPU time limit
# QSUB                  # end of QSUB parameters
cft77 test.f
segldr test.o
a.out
bob> qsub subfile
Request 16.bob submitted to queue: crayque.
bob>
```

The blank # **QSUB** request signals the end of the # **QSUB** section. Current versions of UNICOS will recognize # **QSUB** lines beyond the blank one, but, starting with UNICOS 7.0, such a line will terminate recognition of # **QSUB** lines. Some or all of the values provided on # **QSUB** lines could, instead, have been supplied as options with arguments (and without the # **QSUB**) on the **qsub** command line. Most users find it more convenient to use the # **QSUB** form. In the event that some value is specified on both the **qsub** command line and in a # **QSUB** line, the value on the command line takes precedence.

"Per-process" and "per-request" limits both must be supplied. In general, lower case parameters pertain to per-process limits and upper case, to per-request limits.

A job file submitted via **qsub** is interpreted and executed as a shell script, by default in Bourne shell. It is important that normally other-than-Bourne-shell users provide an alternate (Bourne shell) environment within which the submitted job can execute, or specify that the submitted job is to execute in the desired shell. # **QSUB** **–s** *full-path-name-of-shell* can be used to specify, from within the submitted job, the shell in which the job is to execute.

The **lim** and **limit** commands should not be used in conjunction with NQS jobs because their behaviors are not the same on both machines, and because limits have been imposed on the values which they may specify. Values specified in this manner override earlier # **QSUB** values. On patton only, if **lim** is invoked with a time larger than 10 CPU minutes, the system will terminate the job and send a timeout message, even if # **QSUB** specified a longer time.

8.1.3 Monitoring NQS Jobs

NQS jobs may be monitored with the **qstat** command, which provides 16 options. Three of the more popular usages are:

qstat	displays the owner and status of each queue.
qstat –a	displays the owner and status of each job.
qstat –f *queue*	displays the limits associated with the named queue or subqueue. Nonpipe queue information is more detailed than for a pipe queue.

qdel permits users to delete their own jobs from the queues. If the job is merely queued, **qdel** *job-id* is sufficient. Once the job is running in the queues, **qdel –k** *job-id* is necessary. The man page discusses additional options.

8.1.4 Restrictions

Because of high demand for limited computing resources, the following is ARLSCF policy. A single user may have no more than two jobs per queue or subqueue per machine. A single user may have no more than four NQS jobs running simultaneously per machine. On the Cray X–MP, the deferred queue and all 4 MW (actually, 3.6 MW) queues except debug execute only between 4 PM and 8 AM. Should an extraordinary situation require exceptions to these restrictions, coordination with the system administrator is necessary.

8.2 at, batch, and cron

at permits a user to submit a batch job to start at a certain time. **batch** permits a user to submit a batch job into the MDQS queue. These commands are available on various ones of the ARLSCF computers, but not to the general user community on the Crays.

The **cron** daemon runs continuously on UNIX machines, reading the file **/usr/lib/crontab** once per minute and executing programs as indicated therein. Users do not have write permission to **/usr/lib/crontab**.

In exceptional circumstances, users may discuss their needs to run programs under **cron**, and to use **at** and **batch** on the Crays, with the appropriate system administrators.

8.3 Executing in Background

Suffixing a UNIX command with **&** causes the invoked program to run in background; i.e., asynchronously. When the job is placed in background, the system immediately returns its process id and control to the user, who may then proceed with additional interactive (synchronous) and background activity. The system notifies the user as each background job is completed. The status of background jobs can be checked with the **ps** and **top** commands, and they may be terminated with the **kill** command. Background jobs are terminated at system shutdown, and when the owner logs out. To avoid termination of background jobs upon logging out, users should place the command **nohup** in their **.profile** or **.login** files, or prefix the command with **nohup**:

nohup *command-with-options-and-arguments-and-redirection* **&**

On the Crays, background jobs are considered by the operating system to be interactive, rather than batch, jobs; hence, they are subject to the restrictions of interactive jobs. In particular, such jobs are limited to 10 CPU minutes each on the Cray X–MP.

9. Text Editors

This chapter discusses some of the text editors available on the ARLSCF computers. Extensive on-line documentation is available and is not reproduced here. Users choosing to do their editing on machines other than the Crays will often be rewarded with much more rapid response than the Crays offer.

9.1 Regular Expressions

Text editors typically provide a facility for finding a certain string of characters and then performing some action. For example, in **ed**, **s**/*a*/*b*/**n** substitutes *b* for the first occurrence of *a* on the current line and then prints the line and its line number. In the same manner, *17***s**/*hyperutectic*/*hypereutectic*/**gn** corrects the spelling of every occurrence of *hyperutectic* on line *17* and then prints the line and its line number. **1,$s**/*hyperutectic*/*hypereutectic*/**gn** corrects the spelling of every occurrence of *hyperutectic* throughout the entire file, printing out the text line and line number for every line on which this occurs. Finally, **g**/*hyp.*tic*/**s**//*hypereutectic*/**gn** changes every word beginning with *hyp* and ending with *tic* into *hypereutectic*, printing out the text line and line number for every line on which this occurs. Unfortunately, it also converts the line *He was hyperactive and had a tic.* into *He was hypereutectic.* The strings *a*, *hyperutectic*, and *hyp.*tic* are examples of regular expressions. "." and "*" are examples of regular expression meta-characters, "." meaning any character and "*" meaning any number of occurrences (including 0) of the preceding character.

In the context of UNIX text editors, regular expressions are strings with their own syntax which represent sets of other strings of almost any degree of complexity; thus, they provide UNIX editors with an extremely powerful search capability. Detailed discussions of regular expressions are provided in the Cray publications, *UNICOS User Commands Reference Manual*, SR–2011 6.0 (available on-line; execute **man ed**), and *UNICOS Text Editors Primer*, SG–2050, in the entries for the **ed** editor. The various editors respond to regular expressions in ways which may differ somewhat from each other. These differences are documented.

Figure 9.1 presents some examples of regular expression matches. The regular expressions are shown with their delimiters, and the matching strings are underlined. Comments are italicized.

9.2 ed

ed is the standard UNIX line oriented text editor. Like other line oriented text editors, it tends to be inconvenient for most interactive uses; however, certain text editing tasks may lend themselves particularly well to its use. Large files generate even larger editor temporary files and cost many processor cycles on entry to **ed**. On the Cray–2, the buffer is limited to approximately 17 gigabytes and individual lines, to 4096 characters; reasonable editing sessions should be kept under 10 megabytes. On the Cray X–MP, the buffer is limited to approximately 250 megabytes, and individual lines, to 512 characters. For additional information, execute **man ed** or see the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0. In addition, there is a tutorial in the Cray publication, *UNICOS Text Editors Primer*, SG–2050. Figure 9.2 is a demonstration of **ed**'s use on the ARLSCF Crays. There will be minor (but perhaps disconcerting) differences between the Cray implementation and that on the minicomputers; one such difference is that the minicomputer implementations do not provide **help**. Screen responses/prompts are indented. Every keyboard entry line is followed by a carriage return; in a few cases, • represents an explicit carriage return. "RE" means "regular expression." Comments are

italicized.

*****

/the/	The quick red fox jumped over the lazy brown dog.	
/ /	The_quick red fox jumped over the lazy brown dog.	
/./	The quick red fox jumped over the lazy brown dog.	
/\./	The quick red fox jumped over the lazy brown dog.	
/o./	The quick red fox jumped over the lazy brown dog.	
/o....o/	The quick red fox jumped over the lazy brown dog.	
/o.....o/	The quick red fox jumped over the lazy brown dog.	
/o.........o/	The quick red fox jumped over the lazy brown dog.	
/o.*o/	The quick red fox jumped over the lazy brown dog.	
/.*/	The quick red fox jumped over the lazy brown dog.	
/o.\{4\}o/	The quick red fox jumped over the lazy brown dog.	
/o.\{5\}o/	The quick red fox jumped over the lazy brown dog.	
/o.\{5,\}o/	The quick red fox jumped over the lazy brown dog.	
/o.\{4\}/	The quick red fox jumped over the lazy brown dog.	
/o.\{5\}/	The quick red fox jumped over the lazy brown dog.	
/o.\{5,\}/	The quick red fox jumped over the lazy brown dog.	
/^a*/	aaabbbcccbbbaaa	*constrain to beginning of line*
/a*$/	aaabbbcccbbbaaa	*constrain to end of line*
/^bb*/	aaabbbcccbbbaaa	*one or more b's: no match*
/bb*$/	aaabbbcccbbbaaa	*one or more b's: no match*
/^b*/	aaabbbcccbbbaaa	*zero or more b's: match beginning*
/b*$/	aaabbbcccbbbaaa	*zero or more b's: match end*
/[^a]*/	aaabbbcccbbbaaa	*zero or more not-a's: match beginning*
/[^a][^a]*/	aaabbbcccbbbaaa	*one or more not-a's: match*
/[^ab]*/	aaabbbcccbbbaaa	*zero or more not-a,b's: match beginning*
/[^ab][^ab]*/	aaabbbcccbbbaaa	*one or more not-a,b's: match*
/[ab]*/	aaabbbcccbbbaaa	
/[a–c]*/	aaabbbcccbbbaaa	
/\(a*\)\(b*\)c*\2\1/	aaabbbcccbbbaaa	

Figure 9.1. Some Regular Expressions with Matching Strings

*****

*****

```
ed demo                                  invoke ed, copy file "demo" into buffer
   ?demo                                 ed's responses are indented
   cannot open input file                no such file
help                                     parentheses indicate default line addresses

   (.)a              append until .      P                 toggle prompt
   (.)b[n]           print next screen   q                 quit editor
   (.,.)c            change until .      Q                 force quit
   (.,.)d            delete lines        ($)r [file]       read file
   e [file]          edit new file       (.,.)s/RE/new     substitute RE
   E [file]          force edit new file (.,.)ta           copy after line a
   f [file]          set file name       u                 undo last command
   (1,$)g/RE/cmds    global command      (1,$)v/RE/cmds    inverse global
   (1,$)G/RE/        interactive global  (1,$)V/RE/        inverse interactive
   h                 help on last error  (1,$)w [file]     write file
   H                 toggle the help     X                 enter crypt key
   help              print command help  z                 write and quit
   (.)i              insert until .      /RE               search for string
   (.,.+1)j          join lines          ?RE               search backward
   (.)kx             mark line with x    ($)=              print line number
   (.,.)l            list nonprintables  !command          shell command
   (.,.)ma           move after line a   (.+1)<newline>    print next line
   (.,.)n            numbered print      .                 current line
   N                 switch p and n      $                 last line in file
   (.)o[n]           print overview      ,                 1,$
   (.,.)p            print lines         ;                 .,$

f                                        name of file copied into buffer?
   demo                                  this is also the default file name
.=                                       line number of current line?
   0
=                                        line number of last line?
   0                                     0 because there is nothing in buffer
a                                        append after most recent line
Now is the time
for all good men
to come to the aid of their party.
.                                        exit append mode
w demo.more                              write buffer to file "demo.more"
   68                                    number of bytes
r demo.more                              read into buffer after r's default (last)
   68
2r demo.more                             read into buffer after line 2
   68
.=
   5
1,$pn                                    print buffer lines 1 through last, numbered
   1   Now is the time
   2   for all good men
   3   Now is the time
   4   for all good men
   5   to come to the aid of their party.
   6   to come to the aid of their party.
   7   Now is the time
```

```
   8   for all good men
   9   to come to the aid of their party.
3,5d                                        delete lines 3, 4, 5
e demo.more                                 overwrite buffer with copy of demo.more
   ? warning: expecting 'w'                 reminder to save buffer
e demo.more                                 override
   68
f
   demo.more                                e changed the buffer name
w                                           to default file, demo.more
   68
3m1                                         move line 3 to line after line 1
2,3t0                                       copy lines 2, 3 to top of file
,n                                          print buffer with line numbers
   1   to come to the aid of their party.
   2   for all good men
   3   Now is the time
   4   to come to the aid of their party.
   5   for all good men
1,2d                                        delete lines 1, 2
3t.                                         copy line 3 to line after current line
4d                                          delete line 4
1,$t$                                       duplicate, appending to end
,n
   1   Now is the time
   2   for all good men
   3   to come to the aid of their party.
   4   Now is the time
   5   for all good men
   6   to come to the aid of their party.
3s/the/zzz/n                                substitute 1st occurrence, line 3
   3   to come to zzz aid of their party.
1,$s/the/yyy/n                              substitute 1st occurrence, every line
   6   to come to yyy aid of their party.   only last line prints
3n
   3   to come to zzz aid of yyyir party.   note "their"
1,$s/[yz]/x/gn                              substitute every occurrence, every line
   6   to come to xxx aid of their partx.   note "party"
,n
   1   Now is xxx time
   2   for all good men
   3   to come to xxx aid of xxxir partx.
   4   Now is xxx time
   5   for all good men
   6   to come to xxx aid of their partx.
g/x/s/o/z/g                                 on all lines with x, all "o" to "z"
,n
   1   Nzw is xxx time
   2   for all good men
   3   tz czme tz xxx aid zf xxxir partx.
   4   Nzw is xxx time
   5   for all good men
   6   tz czme tz xxx aid zf their partx.
g/z/d                                       delete all lines with "z"
,n
```

```
   1    for all good men
   2    for all good men
$t$
1,$jn                                          join the lines
   1    for all good menfor all good menfor all good men
s/men/&\                                       split the lines: i.e., replace each occurrence of
/g                                             "men" with "men" and a newline
,n
   1    for all good men
   2    for all good men
   3    for all good men
q                                              exit
   ? warning: expecting 'w'
q                                              repeat forces exit
```

Figure 9.2. An **ed** Demonstration

*****

ed can be used noninteractively by embedding it in a shell script with a "here" file ("here" files are discussed in the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0, under the shell command, **sh**; execute **man sh** for the on-line version). Figure 9.3 shows such a shell script fragment. The fragment is satisfactory only during June. Often, such things are more readily accomplished by other means, among which are the stream editor, **sed**, and the programming language, **awk**. Discussion of **awk** is beyond the scope of this document.

*****

```
ed - <<**********
r !date                    # Fri Jun  9 12:10:43 EDT 1989
s/.....$/,&/               # Fri Jun  9 12:10:43 EDT, 1989
s/^.\{10\}/&,/             # Fri Jun  9, 12:10:43 EDT, 1989
s/  / /                    # Fri Jun 9, 12:10:43 EDT, 1989
s/,.*,/,/                  # Fri Jun 9, 1989
s/^.\{7\}/&e/              # Fri June 9, 1989
s/^.../&day:/              # Friday: June 9, 1989
s/^Tue/&s/                 # Friday: June 9, 1989
s/^Thu/&rs/                # Friday: June 9, 1989
s/^Wed/&nes/               # Friday: June 9, 1989
w mydate                   # into file "mydate"
q                          # quit
**********
```

Figure 9.3. Noninteractive Use of **ed**

*****

9.3 sed

sed, the *stream editor*, can be considered the batch counterpart of **ed**. Unlike **ed**, it can operate on arbitrarily large files because it copies one line into a buffer, executes all the commands which apply thereto (sometimes reading additional lines in the process), writes the result to standard out, and moves on to the

next input line. Many of the editing commands are the same as in **ed**, but there are differences both in the interpretation of the same commands and in the commands available. The editing commands can be presented to **sed** as part of the command line, or in a separate file which will be read by **sed**. Figure 9.4 presents a shell script fragment, using **sed**, which is equivalent to the fragment of Figure 9.3. For additional information, execute **man sed** or see the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0.

*****

```
date | sed -e '{s/.....$/,&/
                 s/^.\{10\}/&,/
                 s/  / /
                 s/,.*,/,/
                 s/^.\{7\}/&e/
                 s/^.../&day:/
                 s/^Tue/&s/
                 s/^Thu/&rs/
                 s/^Wed/&nes/
                }' > mydate
```

Figure 9.4. Use of **sed**

*****

9.4 tr

The translate command, **tr**, is not an editor, but it does perform certain character transformations so nicely that it is well worth addressing here. **tr** is more completely documented in the on-line **man** pages and in the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0. Basically, **tr** changes all occurrences of specified individual characters in a file to other specified individual characters. For example:

tr a b <in >out maps every *a* in file *in* to a *b* and writes the result to file *out*.

tr –s " " " " copies standard input to standard output and, in the process, replaces each string of repeated blanks with a single blank. Note that, in consecutive lines, a trailing string of blanks followed by a leading string of blanks counts as two separate strings because of the intervening newline character.

tr "[a–z]" "[A–Z]" maps every lower case letter into the corresponding upper case letter.

tr "[0–9]" "##########" maps every digit into a #.

tr "[0–9]" "#*10" as above

tr "[0–9]" "#*" as above

tr "[A–Z][a–z][0–9]" "[A*26][a*26][n*]" maps every upper case letter into an *A*, every lower case letter into an *a*, and every digit into an *n*.

tr "\012" "~" maps every newline into a ~. This can be particularly useful prior to manipulating newlines in **sed**. Of course, the inverse **tr** command is executed after **sed**.

tr –cs "[A–Z][a–z][0–9]" "[\012*]" maps all characters except letters and digits into newlines and compresses resulting strings of repeated newlines into single newlines. The result is a list of "words," one per line.

9.5 jove

jove, "Jonathan's Own Version of EMACS," is a widely available screen editor which, by virtue of its capabilities and features, the size of its ARLSCF user community, and the support available at the ARLSCF, has become the preferred editor at the ARLSCF. **jove** hews closely to the conventions of EMACS, although there are some departures. EMACS permits its user to define his own commands in addition to its native set, and also to bind native or created commands to various keys. New commands cannot be created in **jove**, but existing commands can be bound to any key, and commands can be combined into a macro that can be invoked like a standard command.

In common with many other screen editors, **jove** must be informed of the type of terminal from which it is being invoked. This is usually effected upon login, but if the screen seems not to respond properly (more likely when dialing in through a modem), ensure that the following actions have occurred, where *name* is the system's own name for the terminal type:

- in the Bourne shell: **TERM=***name*
export TERM
- in one of the C shells: **set term=***name*

jove is invoked by the command

jove [*options*] [*filenames*]

jove provides one or more windows in which different files, or different parts or the same part of one file, are displayed and manipulated. The position of the cursor on the screen indicates the position within the file, and changes made are shown more or less immediately on the screen. Modifications are made to buffer copies of the files of interest rather than to the files themselves. The buffer copies can replace the original files or be saved as different files in addition to the original files at any time. **jove** attempts to preserve buffer contents in the event of a crash or a loss of communication.

jove has rather a large repertoire of commands, but a surprisingly small subset suffices for most editing tasks. The entire repertoire of commands and their default key equivalents are listed in Figure 9.5. Several commands accept regular expressions (if enabled), and a few prompt the user for additional input; such characteristics are not indicated in the figure. The *variables* permit the user modify the manner in which **jove** responds to certain commands and **CTRL–** key combinations. Every printable character is bound to the self-insert command; thus, merely typing causes text to be entered into the buffer. Except for self-insert, the more frequently used commands initially are bound to various keys:

- depressed in concert with the **CTRL–** key
- prefixed with the Escape key
- prefixed with the Escape key and depressed in concert with the **CTRL–** key
- prefixed with **CTRL–X**
- prefixed with **CTRL–X** and depressed in concert with the **CTRL–** key

*****

Buffer Manipulation

buffer-position	
list-buffers	^X^B
make-buffer-unmodified	Esc ~
select-buffer	^Xb

Deleting

delete-next-char	^D
delete-next-word	Esc d
delete-previous-char	Del, ^H
delete-previous-word	Esc Del; Esc ^H; ^W
delete-to-killbuffer	^X^K
delete-white-space	Esc \
erase-buffer	
kill-buffer	^Xk
kill-to-end-of-line	^K

File Manipulation

find-file	^X^F
find-file-in-other-window	^X4
find-tag	^X^T
insert-file	^X^I
read-file	^X^R
write-current-file	^X^S; ^X^\
write-modified-files	^X^M; ^X^Return
write-named-file	^X^W

Help and Terminal Commands

apropos	Esc h
clear-and-redraw	^L
describe-command	Esc ?
describe-key	^X?
redraw-display	Esc ^L
reinitialize-terminal	
ring-the-bell	^G

Marks

exchange-point-and-mark	^X^X
set-mark	^@; ^(space)

Miscellaneous JOVE Functions

execute-extended-command	Esc x (command)
four-times	^U
next-error	^X^N
parse-C-errors	
parse-LINT-errors	
quote-char	^Q; ^~
source	
string-length	^Xc

Movement (Large)

beginning-of-file	Esc <
beginning-of-window	Esc ,
end-of-file	Esc >
end-of-window	Esc .
goto-line *n*	Esc *n* Esc g
next-page	^V
previous-page	Esc v

Movement (Small)

backward-char	^B
backward-paren	Esc ^B
backward-word	Esc b
beginning-of-line	^A
beginning-of-sentence	Esc a
end-of-line	^E
end-of-sentence	Esc e
forward-char	^F
forward-paren	Esc ^F
forward-word	Esc f
next-line	^N
previous-line	^P
scroll-down	Esc z
scroll-up	^Z

Process Control

exit-jove	^X^C
make	^X^E
pause-jove	Esc p
shell-command	^X!
shell-command-to-buffer	
spell-buffer	
sub-shell	Esc !
suspend-jove (= pause-jove)	

Region Manipulation

append-region	
copy-region	Esc w
filter-region	
write-region	

Searching and Replacing

first-nonblank	Esc m
i-search-forward	^S; ^\
i-search-reverse	^R
query-replace-search	Esc q
replace-search	Esc ^E
search-forward	Esc s
search-reverse	Esc r

Tailoring Jove

bind-macro-to-key	
bind-to-key	
execute-keyboard-macro	
execute-macro	
init-bindings	
name-keyboard-macro	
print (variable)	
read-macros-from-file	
set-quote-chars	
set (variable)	
start-remembering	^X(
stop-remembering	^X)
write-macros-to-file	

Text Modification

c-tab	
case-character-upper	^C
case-region-lower	^X^L
case-region-upper	^X^U
case-word-capitalize	Esc c
case-word-lower	Esc l
case-word-upper	Esc u
character-to-octal-insert	
justify-paragraph	Esc j
newline-and-indent	Newline
newline	Return
newline-and-backup	^O
paren-flash	
self-insert	very bound
text-insert	very bound
transpose-chars	^T

Undeleting

yank	^Y
yank-pop	Esc y

Windows

delete-current-window	^Xd
delete-other-windows	^X1
grow-window	^X^
next-window	^Xn
number-lines-in-window	
page-next-window	Esc ^V
previous-window	^Xp
shrink-window	
split-current-window	^X2

Variables

allow-^S-and-^Q	
auto-indent	
backup-files	
c-mode	
case-independent-search	
fast-prompt	
files-should-end-with-newline	
internal-tabstop	
make-all-at-once	
overwrite	
physical-tabstop	
regular-expressions	
right-margin	
show-match	
scroll-step	
text-fill	
visible-bell	
write-files-on-make	

Notes: ^ alone is the character "^"; prefixing another character, it means "CTRL–".
n means an integer.
The space which appears after Esc is for legibility only.
Some commands (e.g., select-buffer) prompt the user for a value.

Figure 9.5. jove Commands and Cray Default Bindings

*****

The following commands are probably the most frequently used ones, and are sufficient to do a considerable amount of text editing:

- **Forward-character** and **backward-character** (^f, ^b; "^" represents "CTRL–") move forward and backward one character position in the buffer. At the end of each line of text is a newline character; moving forward across it advances the current position to the beginning of the next line.
- **Forward-word** and **backward-word** (Esc–f, Esc–b) are similar to forward-character and backward-character, but move by words (continuous strings of alphanumeric characters).
- **Next-line** and **previous-line** (^n, ^p) move by lines.
- **Delete-next-character** and **delete-previous-character** (^d, ^h) delete the character immediately before and after the current position.
- **Delete-next-word** and **delete-previous-word** (Esc–d, Esc–h) are similar to delete-next-character and delete-previous-character, but delete words or partial words to either side of the current position.
- Escape, followed by a number, followed by a command, executes that command that number of times. For example, Esc–10^n moves down 10 lines.
- **Set-mark** (^@) marks one end of a region, the other end being the current position. Regions, like characters, words, and lines, are the operands of various commands.
- **Delete-to-killbuffer** (^x^k) removes all between the mark and the current position, saving it in the killbuffer.
- **Yank** (^y) inserts a copy of the killbuffer contents at the current position.
- **Incremental-search-forward** and **incremental-search-reverse** (^\, ^r) search forward and backward to the next occurrence of the specified string. Accepts regular expressions, if they are enabled.
- **Search-forward** and **search-reverse** (Esc–s, Esc–r) prompt for and then search forward and backward to the next occurrence of the specified string. Accepts regular expressions, if they are enabled.
- **Query-replace-search** (Esc–q) prompts for a search string and replacement string, and then searches for the search string, replacing selected occurrences with the replacement string. At each occurrence, the user is prompted and can respond "yes", "no", "all", "exit", or "recursive". Accepts regular expressions, if they are enabled.

For additional information execute **man jove**.

9.6 vi

vi is the standard UNIX screen (visual) text editor. **ex** is an underlying line oriented editor for **vi**. Its commands, preceded by a colon and followed by a carriage return, may be used from within **vi**, and are often the only way to accomplish certain tasks within **vi**. Because of **jove**'s preeminent position at the ARLSCF, little support is available to the user wishing to use **vi**, and it is emphasized that **jove** is the screen editor of choice at the ARLSCF.

In common with many other screen editors, **vi** must be informed of the type of terminal from which it is being invoked. This is usually effected upon login, but if the screen seems not to respond properly (more likely when dialing in through a modem), ensure that the following actions have occurred, where *name* is the system's own name for the terminal type:

- in the Bourne shell: **TERM=***name*
export TERM
- in one of the C shells: **set term=***name*

vi is invoked by the command

vi [*options*] [*filenames*]

or, for a special verbose mode particularly helpful to the beginner,

vedit [*options*] [*filenames*]

The position of the cursor on the screen indicates the position within the file, and changes made are shown more or less immediately on the screen, but are made only to a buffer copy of the file of interest. The buffer copy is saved upon the user's command. **vi** attempts to preserve buffer contents in the event of a crash or a loss of communication.

vi has rather a large repertoire of commands, many of which are listed in Figure 9.6. That figure is intended as a quick reference summary; space therein prevents the listing of all the details. For example, "dw" is described as "delete word." It is not stated that this means "delete to the right from the current position through the end of the current word, including trailing white space, not including trailing punctuation, but if the current position is white space, delete from the current position through the end of the white space." For additional information execute **man vi** and **man ex**, or see the Cray publications, *UNICOS User Commands Reference Manual*, SR–2011 6.0, and *UNICOS Text Editors Primer*, SG–2050.

Initially, and upon return from insert and **ex** modes, **vi** is in command mode. Insert mode is selected by entering one of the characters **aiAIoOcCsSR**. It permits the entry of arbitrary text and is terminated normally with Esc or abnormally with ^?. **ex** mode is entered with **Q** and permits the use of **ex** commands as if **ex** had been invoked, rather than **vi**; in addition, **ex** commands may be entered individually from command mode by pre- and postfixing each one with **:** and RETURN, respectively. **vi** supports string searches and substitutions using regular expressions. **vi** *options* permit the user to modify the manner in which **vi** responds to certain commands and **CTRL–** key combinations.

*****

Adjusting the Screen

^L; ^R	redraw; eliminating @ lines
z•; z–; z.	redraw, current line at top; bottom; center
z*n*.	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

File Manipulation

:e!•; :e *f*•	start over; edit file *f*; changes not saved
:e #•	return to previous file, changes not saved
:q•; :q!•	quit; quit, discard changes
:w•; :w *f*•; :w! *f*•	write edit buffer; to file *f*; overwrite *f*
:x•	write edit buffer then exit vi
:n•	edit next file in arglist
:n *args*•	specify new arglist
ZZ	write edit buffer then exit vi

Inserting/Modifying/Deleting Text

a*text*Esc	append *text* after cursor
A*text*Esc	append *text* at end of line
cw*text*Esc	*text* replaces current position to end of word
C*text*Esc	*text* replaces current position to end of line
:*i*,*j*d•	delete lines *i* through *j*
*n*dd	delete *n* lines starting with current line
d´*x*	delete current line through line marked *x*
dw	delete current character to end of word
D	delete current character to end of line
i*text*Esc	insert *text* before cursor
I*text*Esc	insert *text* before first nonwhite
J	join line and next line, intervening space
:*i*,*j*m*k*•	move lines *i* through *j* to just after *k*
o*text*Esc	open new line below, insert *text*
O*text*Esc	open new line above, insert *text*
p; "*n*p; "*x*p	put buffer text after cursor or current line
P; "*n*P; "*x*P	put buffer text before cursor or current line
r*x*	*x* replaces current character
R*text*Esc	*text* replaces characters, one for one
s*text*Esc	*text* replaces current character
S*text*Esc	*text* replaces current line
u	undo most recent change to edit buffer
U	undo most recent change to current line
x	delete current character
X	delete preceding character
y	yank current character
yw	yank current word
yy; Y	yank current line
y´*x*	yank current line through line marked *x*
:*i*,*j*y•	yank lines *i* through *j*
<<; >>	left shift by tabs; right shift by tabs
.	repeat most recent edit buffer changing cmd

While Inserting/Modifying Text

^H	erase preceding character
^W	erase current word
^U	erase to beginning of insert
\	quote ^H, ^W, ^U
Esc	end insertion, back to command mode
^?	interrupt, terminate insert
^D	backtab over autoindent
0^D	kill autoindent for rest of file
u_arrow^D	kill autoindent for current insert
^V	quote nonprinting character

Miscellaneous

Esc	cancel incomplete command
^C	interrupt command in progress
^?	interrupt
^G	show current file and line
\	quotes the special characters

Movement (Large)

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
H	1st nonwhite in top line on screen
L	1st nonwhite in last line on screen
M	1st nonwhite in middle line on screen
*n*G	1st nonwhite in line *n* (default: last)
{	preceding beginning of paragraph
}	next beginning of paragraph
[[	beginning of file
]]	end of file

Movement (Small)

+; •	1st nonwhite in next line
–	1st nonwhite in previous line
d_arrow; j	next line, same column
u_arrow; k	previous line, same column
n|	to column *n*, default 1
0	beginning of current line
$	end of current line
^	first nonwhite in current line
r_arrow; l; (space)	forward one character
l_arrow; h; ^H	backward one character
w	next beginning of word or punctuation
b	preceding beginning of word or punctuation
e	next end of word or punctuation
W	next beginning of word
B	preceding beginning of word
E	next end of word
(	preceding beginning of sentence
)	next beginning of sentence

Searching and Substituting

/*string*•	next occurrence of *string*
?*string*•	preceding occurrence of *string*
/*string*/+*n*•	*n*th line after *string*
?*string*?–*n*•	*n*th line before *string*
:*i*,*j*s/*s1*/*s2*/g•	global substitute *s2* for *s1* in lines *i* through *j*
n	repeat most recent / or ?
N	repeat most recent / or ?, reverse direction
f*x*	cursor to next *x* in line
F*x*	cursor to preceding *x* in line
t*x*	cursor before next *x* in line
T*x*	cursor after preceding *x* in line
;	repeat last f, F, t, or T
,	reverse last f, F, t, or T
%	find matching (,), {, or }

Marking and Returning

m*x*	mark current position with *x*
`*x*	go to mark *x*
´*x*	go to first nonwhite in *x*-marked line
``	toggle position between mark and other
´´	toggle position between beginning of marked line and other

Escape to Shell

:sh•	open subshell
:!*cmd*•	run *cmd*, return
:r!*cmd*•	run *cmd*, insert output, return

Options (set command)

autoindent	continue previous indentation (insert mode)
autowrite	write before changing files
ignorecase	ignore case when searching
lisp	(), {} are s-expressions
list	display ^I for tab, $ at end of line

set	modify **vi**'s behavior	magic	turn on metacharacter meaning of ., [, *
Q	enter **ex** mode	noai	turn off autoindent
.	addresses current line	noic	turn off ignorecase
$	addresses last line	nonumber	turn off line numbering
%	addresses all lines	number	number lines
		redraw	redraw screen
		report	threshold for number of lines modified
		scroll	command mode lines
		shiftwidth	set reverse tabbing stop
		showmatch	show the match to) and { when typed
		showmode	show insert mode in vi
		slowopen	stop updates during insert
		window	visual mode lines
		wrapscan	regular expression search wraps around file
		wrapmargin	set margin for text wrap

Notes: ^ alone is the character "^"; with another character, it means "CTRL–".

n means an integer, *s* a string, *x* a letter.

• represents a carriage return.

Many commands accept numerical prefixes, with the meaning of line number, range of line numbers, column number, number of lines to scroll, or number of times to repeat the effect of a command.

Line numbers often may be replaced by line numbers relative to the current line (+ or – number), by search strings, or by line markers prefixed with single apostrophes.

Delete and yank (copy) commands do so into the current buffer, whence the text can be recovered. In addition, **vi** automatically creates nine more such buffers, named 1 through 9, for the nine most recent deletions, the most recent being in 1. In addition, the user can create 26 more such buffers named lower case "a" through "z" by prefixing the delete or yank command with "*letter* (e.g., "q5dd). Prefixing a "p" or "P" command with "*letter-or-number* (e.g., "qP) retrieves the contents to the edit buffer.

Figure 9.6. vi Commands

*****

10. cf77 Compiling System

The entirety of this chapter applies only to the ARLSCF Cray computers.

cf77 is described as a compiling system because it incorporates processing phases before and after the compiler proper (**cft77**). All the phases except the loader can be said to "compile" Fortran source code, but only the compiler proper produces binary output, which is then processed by the loader to produce an executable program. It is two preliminary phases of the **cf77** process that may produce significant improvement in a program's performance. The first one, **fpp**, analyzes a program to detect dependencies and adds directives for use in later phases. The second one, **fmp**, translates **fpp**'s output (still essentially Fortran) to enable multitasking, or parallel processing. This automated multitasking capability is called "autotasking."

The action of the loader is suppressed by several **cf77** options, in which event the relocatable binary output corresponding to each input file (see *files* at the end of the **cf77** options list, following) is default named with the same root name and a **.o** suffix, or as specified by still another option. When the loader is not suppressed, the executable binary output is default named **a.out**, or as specified by a **cf77** option or a **segldr** option or directive.

The general preprocessor, **gpp**, is an optional phase of the **cf77** system which performs functions formerly done by the C preprocessor, **cpp**. These include macro substitution, conditional compilation with directives **#if**, **#ifdef**, and **#ifenddef**, and use of **#include** files. **gpp** is preferable to **cpp** because its line numbering scheme is compatible with the other **cf77** processing phases. **gpp** is invoked automatically for input files suffixed with **.F** instead of **.f**.

Of itself, the compiler proper (**cft77**) does not provide the enhanced optimization capabilities of **fpp** and **fmp**, nor the convenience of **gpp** and of automatic invocation of **segldr**.

Throughout this chapter, a number of commands with options, options with arguments, and arguments are presented. In general, typical UNIX conventions concerning white space, quoting, and metacharacters apply.

Detailed information may be found in the on-line manual pages and in the Cray publications, *CF77 Compiling System*,

Volume 1: Fortran Reference Manual, SR–3071 5.0
Volume 2: Compiler Message Manual, SR–3072 5.0
Volume 3: Vectorization Guide, SG–3073 5.0
Volume 4: Parallel Processing Guide, SG–3074 5.0

and *Segment Loader (SEGLDR) and ld Reference Manual*, SR–0066 6.0, and *UNICOS Performance Utilities Reference Manual*, SR–2040 6.0.

10.1 cf77: Syntax and Options

cf77 [**–Z** *phase*] [**–Wd"***string***"**] [**–Wu"***string***"**] [**–Wf"***string***"**] [**–Wa"***string***"**] [**–Wl"***string***"**] [**–Wp"***string***"**] [**–c**] [**–C**[*cpu*][,*hdw*]...] [**–F**] [**–g**] [**–G**] [**–I** *incldir*] [**–J**] [**–l** *lib*] [**–L** *dir*[,*dir*]...] [**–M**] [**–N** *col*] [**–o** *outfile*] [**–S**] [**–T**] [**–v**] [**–V**] [**–D** *name*[=*def*]] [**–P**] [**–U** *sym*] [**––**] *files*

–Z *phase* Specifies the phases of the **cf77** compiling system to be invoked. *phase* specifies a code generation option. The default is **c**, activating only the compiler and loader. Arguments **p**, **u**, and **v** can be specified as **P**, **U**, and **V**, in order to save intermediate files from the initial phases of **cf77**.

- **p** Activates the entire compiling system. **gpp** processing is invoked or not depending on the input file name suffix, **.f** (not invoked) or **.F** (invoked).
- **v** Activates all phases but **fmp**.
- **u** Activates all phases but **fpp**.
- **c** Activates the compiler and loader only (default).
- **m** Specifies use of premult (to be removed in **cf77** 6.0).

–W*key*"*string*" Passes a quoted string containing options to the compiling system phase selected by the *key* letter. Options are expressed in the same format that would be used if that compilation phase were being explicitly invoked by its own command.

key	phase	command
p	generic preprocessor	**gpp**
d	dependence analyzer	**fpp**
u	parallel processing translator	**fmp**
f	Fortran compiler	**cft77**
a	assembler	**as**
l	segment loader	**segldr**

–c Disables the load step and saves the binary file with a **.o** ending. The default is to load and then delete the binary file.

–C[*cpu*][,*hdw*]... Specifies mainframe and hardware for which the program is to be compiled. Use of this option is not recommended for alternate *cpu* choice. Instead, use the **TARGET** environmental variable. See the manuals or execute **man target** for additional information.

–D *name*[=*def*] Interpreted only by **gpp**. Defines *name* as by a **#define** directive. If there is no =*def*, *name* is defined as **1**. Up to 64 names can be specified, each with its own **–D**.

–F Activates the flowtrace feature which permits monitoring of the program during execution. Generates the **flow.data** file which is read by the **flowview** utility.

–g Generates a debug symbol table; equivalent to **–Wf"–ez –o off"**; all **–Z** options are ignored.

–G Generates a debug symbol table with no effect on optimization; equivalent to **–Wf"–ez"**, or when **–Zv** is used, **–Wf"–ez" –Wd"–dc –ef"**.

–I *incldir* Names additional directories to be searched in left to right order for files specified by relative names in **INCLUDE** lines. The directory containing the input file is searched first and is the default. Up to 10 directories can be specified, each with its own **–I**. When used with **gpp**, **–I** applies to the **#include** files with relative names. Thus, **#include** files whose names are enclosed in "" are first searched for in the directory of the input file, then in directories specified by **–I**, and finally in the standard directories. For **#include** files enclosed in <>, the directory of the input file is not searched.

–J Processes Fortran programs through **fpp** and/or **fmp** without compilation and creates output files with **.j** or **.m** suffix, respectively. **–Zp** or **–Zu**, respectively, also must be specified.

–l *lib* Identifies library files. If the library name begins with . or /, it is used without modification; else, directories in the directory search list (default: **/lib**, **/usr/lib**) are searched for file **lib***name***.a**.

–L *dir*[,*dir*]... Prepends the directory search list with the named directories.

–M Processes the Fortran program through **fpp** and leaves the modified files with a **.m** suffix. If **–Z** is not specified, **–Zp** is assumed.

–N 72|80 Specifies **72**- or **80**-column wide source code lines. Default: **72**.

–o *outfile* Overrides default output file name **a.out**. This option is interpreted by the loader and does nothing if binary is saved by **–c** or by **–Wf"–b** *file*".

–P Only **gpp** is invoked, and output is placed in *file*.**i** (root name from *file*.**F**).

–S Generates Cray Assembly Language (CAL) output in *file*.**s** (root name from *file*.**f**); equivalent to **–Wf"–eS"**.

–T Disables the entire compiling system but displays all options currently in effect for each system phase. Same as **–v** option but with no processing.

–U *sym* Interpreted only by **gpp**. Removes any initial definition of reserved symbol name *sym* that is predefined to **gpp**, as by an **#undef** directive. Up to 64 symbols can be specified, each with its own **–U**.

–v Indicates each phase of the compilation system as it is encountered and displays its options and arguments. Output written to stderr (typically the screen), not to stdout or the listing file.

–V Activates an option of the same name on each phase of the compiling system. The compiler reports compilation time and statistics, and system version. Information is sent to stderr (typically the screen) and to the listing file.

–– Signifies the end of the options; input file names follow.

files Names of input files containing Cray Fortran source code. Names must be suffixed **.f** or **.F**. The **.F** suffix indicates that the file is to be preprocessed by **gpp** (if active).

10.2 cf77: Environmental Variables

The following environmental variables are a part of the execution environment and affect the **cf77** system:

CAL Contains the file name for the CAL assembler; default is **/bin/as**.

CFT77 Contains the file name for the Fortran 77 compiler; default is **/bin/cft77**.

FMP Contains the file name for the **fmp** translator; default is **/bin/fmp**.

FPP Contains the file name for the **fpp** dependence analyzer; default is **/bin/fpp**.

SEGLDR Contains the file name for the segment loader; default is **/bin/segldr**.

NCPUS Contains the number of CPUs available to work on a program; default is the number of physical CPUs on the machine.

NPROC Contains the number of processes used for simultaneous compilations; default is **1**.

PREMULT Contains the file name for the microtasking preprocessor (to be removed in **cf77** 6.0); default is **/bin/premult**.

10.3 cf77: Default Files

The input, output, and intermediate files used by **cf77** follow. *file* is the user-selected root name.

file.**a**	Library file.
file.**f**	Fortran source file.
file.**F**	Fortran source file to be preprocessed by **gpp**.
file.**s**	Assembly language file.
file.**o**	Object file.
file.**j**	Output file from **fmp**.
file.**l**	Fortran listing file.
file.**m**	Output file from **fpp**.
a.out	Default name of executable output file.

10.4 cft77 Compiler

Correct use of **cf77** requires an understanding of **cft77**, which is the compiler proper invoked by **cf77**, and which can be invoked explicitly. Options specified below are presented to **cf77** as *options* within **–Wf**"*options*", or they may be presented to an explicit invocation of **cft77**. Use of **cf77** is recommended, rather than explicit invocation of **cft77**.

10.4.1 cft77: Syntax and Options

cft77 [**–a** *alloc*] [**–A** *adrmode*] [**–b** *binfile*] [**–c** *cifopts*] [**–C** *cpu,hdw*] [**–d** *offstring*] [**–D** *cdirlist*] [**–e** *onstring* [**–i** *intlen*] [**–I** *inlname*] [**–l** *listfile*] [**–m** *msglev*] [**–M** *msglist*] [**–N** *col*] [**–o** *optim*] [**–P** *incldir*] [**–R** *runchk*] [**–s** *calfile*] [**–t** *trunc*] [**–V**] *files*.**f**

Defaulting on all the options is equivalent to:

cft77 –astatic –Afull –b*files*.**o –C***host–characteristics* **–dacdfghijmnoPsSuvwxz –eBpqr –i46 –m3 –N72 –o noagress,bl,noinline,nokernsched,noloopalign,recurrence,norecursive,scalar,vector, vsearch,nozeroinc** (normal UNIX continuation of preceding line) **–P***additional-INCLUDE-directories files*.**f**

–a *alloc* Selects memory allocation scheme **static** or **stack**. **static** causes all memory to be statically allocated; that is, any variable allocated to memory occupies the same address throughout program execution. If **stack** is selected, read-only constants and variables in **DATA** and **SAVE** statements, and **COMMON** blocks, are **static**; all other variables are **stack** (their storage location may be used by other entities). Default: **static**.

–A *adrmode* elects addressing mode **full** or **fast** on a Cray X–MP EA system equipped with extended memory hardware. Default: **full**.

–b *binfile* Creates file *binfile* for the binary load modules. Disabled by **–dB**, **–eS**, and **–s**. Default: *file*.**o**, root from *file*.**f**.

–c *cifopts* Creates compiler information file, *file*.**T** (root from *file*.**f**), containing information for use with programming tools which will be available in UNICOS 7.0. See the Cray publication, *Compiler Information File (CIF) Reference Manual*, SM–2401 1.0, for detailed information. Default: none.

–C [*cpu*] [,*hdw*]... Same as the **–C** option for **cf77**, but effective only during compilation. See the manuals or execute **man target** for additional information. Default: host characteristics.

–d *offstring* Disables the compiler functions specified in *offstring*. These compiler functions are listed in the following section. Default: **–dSacdfghijmnosuvxz**; Cray–2 also **Pw**.

–D *cdirlist* Disables comma-separated compiler directives in *cdirlist*. Default: all enabled except **INLINE** and **NOINLINE** are disabled unless inlining is activated; see **–I** and **–o**, following.

–e *onstring* Enables the compiler functions specified in *onstring*. These compiler functions are listed in the following section. Default: **–eBpqr**

–i 46|64 Specifies **46**- or **64**-bit integer arithmetic. Default: **46**.

–I *inlname* Activates explicit inline code expansion within programs contained in file or directory *inlname*. Default: not activated.

–l *listfile* Creates file *listfile* to receive listing output enabled by **–e** options **c**, **g**, **m**, **s**, **x** and by the **LIST** and **CODE** directives. Default: *file*.**l**, from *file*.**f**.

–m *msglev* Specifies the lowest level messages to be issued. Message levels are:

- **0** Comments
- **1** Notes
- **2** Caution messages
- **3** Warning messages, default
- **4** Error messages

–M *msglist* Disables messages by comma-separated message numbers in *msglist*. Default: none.

–N 72|80 Specifies **72**- or **80**-column wide source code lines. Default: **72**.

–o *optim* [,*optim*]... Specifies code optimizations to be performed during compilation. Optimization codes are listed in the second following section.

–P *incldir* Names additional directories to be searched in left to right order for files specified by relative names in **INCLUDE** lines. The directory containing the input file is searched first and is the default. Up to 20 directories can be specified, each with its own **–P**.

–R *runchk* Specifies run-time checks. *runchk* is a concatenation of the desired ones of:

- **a** Compare number and type of arguments passed to procedures with number and type expected.
- **b** Check subscripts against array bounds.
- **c** Check conformance of arrays in array expressions.

Default: none.

–s *calfile* Places CAL output in file *calfile*. Suppresses loadable binary output. Default: none, or *file*.**s** with **–eS**.

–t *trunc* Replaces rounding of arithmetic results with truncation of the least significant *trunc* bits; *trunc* < 48. Double-precision variables, function results, and constants are not affected. Default: rounding.

–V Provides compilation summary information to file stderr and to listing file if any. Default: information not provided.

`--`	End of options.
files.**f**	Names of the source files. Required.

10.4.1.1 Arguments for –d and –e, Disable and Enable

Option	Default	Description
B	**–e**	Creates binary object file; no CAL file created.
P	**–d**	Cray–2 only; local memory paging.
S	**–d**	Generates CAL file to *file*.s.
a	**–d**	Aborts compilation after first fatal error.
c	**–d**	Permits common block cross reference listing.
d	**–d**	Generates debug table; **–ez** preferred.
f	**–d**	Generates flowtrace output; **cf77 –F** preferred.
g	**–d**	Generates listing of binary output with CAL equivalent.
h	**–d**	Lists only the first statement and the error messages in each program.
i	**–d**	Causes a runtime error when an uninitialized local variable is used in a floating point operation or in array subscripting.
j	**–d**	Executes at least one iteration of DO loop if its DO statement is executed.
m	**–d**	Enables loopmarked listing of source code to listing file; see –l, preceding.
n	**–d**	Generates messages for all nonstandard usages.
o	**–d**	Generates code for runtime checks of array conformance and subscripts versus array bounds; equivalent to **–Rbc**; to be removed in **cft77** 6.0.
p	**–e**	Allows double precision. **–dp** causes double precision entities to be compiled as single precision, and double complex entities (which otherwise cause a fatal compilation error) to be compiled as complex.
q	**–e**	Aborts compilation when 100 fatal errors are encountered.
r	**–e**	Rounds multiplication results, overriding **–t**. Cannot be disabled on Cray–2.
s	**–d**	Enables listing of source code to listing file; see –l, preceding.
u	**–d**	Permits special rounding of real division results.
v	**–d**	Causes all variables in all program units to be treated as though they appeared in a **SAVE** statement.
w	**–d**	Cray–2 only; to be removed in **cft77** 6.0; same as **P** argument.
x	**–d**	Permits cross reference listing.
z	**–d**	Permits use of debugging tools by generating debug table.

10.4.1.2 Arguments for –o, Optimization

The optimization option has form **–o***list*, where *list* is a comma-separated list of desired arguments. The arguments exist as pairs, to enable or disable various features. Each pair except one is of form *feature* and **no***feature*. The exception is **on**, **off**.

aggress	Causes the compiler to increase certain internal limits, thus allowing some loops to be vectorized which otherwise would not have been. Default: **noaggress**.
bl	Permits full bottom loading of scalar operands in loops. Default: **bl**.
inline[*n*]	Activates automatic inline code expansion for called subprograms. *n* is **1** (default), **2**, or **3**, indicating the number of levels to be inlined. Default: **noinline**.
kernsched	Cray–2 only. Causes compiler to use polycyclic scheduling for small kernel loops to increase parallelism. Default: **nokernsched**.
loopalign	Causes the compiler to attempt to align **DO** and **DO WHILE** (nonstandard) loops on buffer boundaries to decrease overhead. Default: **noloopalign**.

recurrence Permits the vectorization of reduction loops. Default: **recurrence**.

recursive Causes the compiler to assume that all subprograms have direct or indirect recursion. Default: **norecursive**.

scalar Permits scalar optimization and associated directives. Default: **scalar**.

vector Permits vectorization and associated directives. Default: **vector**.

vsearch Permits vectorization of search loops. Default: **vsearch**.

zeroinc Causes the compiler to assume that constant increment variables might be incremented by 0, requiring generation of conditional vector code. Default: **nozeroinc**.

off Turns off all optimization. Default: **on**.

10.4.2 cft77: Compiler Directives – CDIR$

Compiler directives provide a means to enable and disable particular compilation features from within the source code; thus, they provide a means for limiting features to portions of subprograms. Compiler directive lines are identified by **CDIR$** in columns 1 through 5 and a blank in column 6; thus, they appear to other Fortran compilers as comments. The directive itself occupies columns 7 through 72 (or 7 through 80 with **–N80**). A line may contain multiple directives, except that directives with lists must be on lines by themselves. With the exceptions of **EJECT**, **LIST**, and **NOLIST**, directives must not appear outside of a program unit. Unless otherwise stated, the effect of a directive terminates when the end of a program unit is encountered, or when a directive with the opposite sense is encountered. *list* indicates a comma-separated list of the entities of interest.

10.4.2.1 Output Directives

CODE/NOCODE Similar to **LIST/NOLIST** but pertains to the generated binary object code (not CAL listing). The last directive encountered controls the entire program unit.

EJECT Inserts a page break at the corresponding point in the otherwise enabled source listing.

LIST/NOLIST Starts and stops source listing. Overrides command line options.

10.4.2.2 Vectorization Directives

IVDEP[SAFEVL=*n*] Causes the compiler to ignore vector dependencies when the vector length is at least *n*, default all vector dependencies. Applies to the first **DO** or **DOWHILE** loop, or array syntax assignment, following the directive and in the same program unit.

NEXTSCALAR Suppresses vectorization of following DO or DO WHILE (nonstandard) loop.

RECURRENCE/NORECURRENCE Toggle on and off otherwise enabled vectorization of reduction loops; directives override **–o recurrence** but not **–o norecurrence**. A reduction loop reduces an array to a scalar value by doing a cumulative operation on all of the array's elements; this involves including the result of the previous iteration in the expression of the current iteration.

SHORTLOOP Declares that a loop has a trip count less than 64, thereby eliminating the need for code to test for completion of that loop. Applies to the first **DO** or **DOWHILE** (nonstandard) loop following and in the same program unit as the directive. Effective only in vectorized loops.

VECTOR/NOVECTOR Toggles on and off otherwise enabled vectorization.

VFUNCTION *list* Indicates that the listed external functions have vector versions.

VSEARCH/NOVSEARCH Toggles on and off otherwise enabled vectorization of search loops. A search loop is a loop which can be exited by means of an **IF** statement.

10.4.2.3 Scalar Optimization Directives

ALIGN Aligns a block of code on an instruction buffer boundary. Aligning dominant loops which can fit within one instruction buffer can decrease overhead.

BL/NOBL Toggles between full and safe bottom loading within loops for which bottom loading is otherwise permitted.

NO SIDE EFFECTS *list* Declares that listed subprograms do not redefine their arguments, variables in common blocks, and variables local to their calling program. Permits register storage across subprogram calls. Not needed for **INTRINSIC**s and **VFUNCTION**s.

SUPPRESS [*list*] Suppresses scalar optimization at the point where the directive occurs. Variables in registers are stored to memory, whence they are read at their next reference. *list* restricts the effect of the directive to the named arrays.

10.4.2.4 Storage Directives

AUXILIARY *list* Allocates listed arrays and common blocks to the solid-state storage device. Not available at the ARLSCF.

REGFILE [*list*] On the Cray–2 only, assigns named common blocks to local memory. The directive must occur before the first executable statement in the program unit.

STATIC/STACK Select **STATIC** or **STACK** allocation for local data. Override **–a**. The last occurrence within a program unit is effective for the entire unit.

10.4.2.5 Other Directives

BOUNDS [*list*]/**NOBOUNDS** [*list*] Toggle on and off the checking of subscripts against array boundaries. Appear after all specification statements. Override **–Rb**. *list* restricts the effect of the directive to the named arrays.

FLOW/NOFLOW Turn flowtrace on and off. Override **–ef** and **–df**. The last occurrence within a program unit is effective for the entire unit.

INLINE/NOINLINE Toggle on and off inline code generation when inlining is enabled by **–I** or **–o inline**.

INTEGER=46|64 Selects 46- or 64-bit integers. Overrides **–i**. The last occurrence within a program unit is effective for the entire unit.

TASKCOMMON *list* Declares the listed named common blocks to be local to a single multitasking task. Makes independent copies of listed blocks used by more than one task. The directive must occur before the first executable statement in the program unit, and common blocks so identified in any program unit must be so identified in all program units wherein they appear.

10.5 segldr Loader

segldr is the loader invoked by **cf77**. Its options, specified in the following section, are presented to **cf77** as *options* within **–Wl**"*options*", or they may be presented to an explicit invocation of **segldr**. Use of **cf77** is recommended, rather than explicit invocation of **segldr**.

Beginning at the main program in the calling tree, **segldr** links together the needed relocatable binaries to produce an executable binary. The relocatable binaries are selected first from the object (**.o**) files named in *objfiles*, and then from user-supplied libraries (–l option), and then from the default libraries. The object files may have been generated by the compilers (e.g., **cft77**, **cc**) or the assembler, or have been extracted from libraries by **bld** or **ar** (**bld** is recommended). User libraries are searched in the order in which they are listed. Directories named in the directory search list (**–L** option; defaults are **/lib** followed by **/usr/lib**) are searched in order for user-specified libraries and directives files whose names have not been specified with a leading . or /, and for default libraries. In the event of multiple occurrences of needed entry point names, the first one encountered is accepted, the others being ignored. The default library set is: **libc.a**, **libf.a**, **libio.a** (Cray X–MP only), **libm.a**, **libp.a**, **libsci.a**, and **libu.a**.

10.5.1 segldr: Syntax and Options

segldr [**–a**] [**–A** *file*] [**–b** *value*] [**–D** *dirstring*] [**–e** *name*] [**–E**] [**–f** *value*] [**–F**] [**–g**] [**–H** *hi*[+*he*]] [**–i** *dirfiles*] [**–j** *names*] [**–k**] [**–l** *names*] [**–L** *ldirs*] [**–m**] [**–M** *arguments*] [**–n**] [**–N**] [**–o** *outfile*] [**–O** *keyword*] [**–s**] [**–S** *si*[+*se*]] [**–t**] [**–u** *unames*] [**–V**] [**–z**] [**–Z**] *objfiles*

–a Aligns all code blocks and local data blocks on instruction buffer boundaries.

–A *file* Identifies *file* as an existing executable to which relocatables and library extractions will be linked.

–b *n* Adds *n*, 1024 word blocks to the BSS space of the loaded program.

–D *list* Provides a semicolon-separated list of **segldr** directives. **segldr** processes these directives before any directives files.

–e *ename* Indicates that program execution is to begin at entry *ename*.

–E Echoes to the load map all directives processed.

–f *keyword* Sets all uninitialized data to one of the following *keyword* selected values, or to the actual *value* specified.

- **zeros** All bits set to 0 (default).
- **ones** All bits set to 1.
- **indef** 0605054000000000000000 octal, to cause a floating-point error if used in a floating-point operation.
- **–indef** Same as **indef**, but with the leading bit on.
- **indefa** Result of a logical OR operation of 0605054000000000000000 octal and the address of the word being preset.
- **–indefa** Same as **indefa**, but with the mask's leading bit on.
- *value* *value*, where *value* is a 16-bit octal number between 0 and 177777, inclusive; *value* is replicated four times in the 64-bit word.

–F Forces the loading of all modules in the named object files, even if not referenced.

–g Generates the debug symbol tables and appends them to the executable file. Enabled by default, disabled by **–s**.

–H *hi*[+*he*] Assigns the initial heap size (*hi*) and the heap expansion increment (*he*).

–i *list* Causes the directives in each of the directives files in *list* to be read. – represents stdin.

–j *list* A comma-separated list of directives file names. A name beginning with "." or "/" is assumed to be a complete path name; otherwise, **segldr** looks for file **segdirs/***name* in the search directories.

–k Redirects all but summary class error messages to the load map file.

–l *list* A comma-separated list of library file names. A name beginning with "." or "/" is assumed to be a complete path name; otherwise, **segldr** looks for file **lib***name***.a** in the search directories.

–L *list* Adds the listed directories to the beginning of the directory search list. Not for the faint hearted. Default list is **/lib**, then **/usr/lib**.

–m Same as **–M,a**.

–M [*file*] [,*opts*] Names the load map file for paginated 132-column output (default: stdout, unpaginated, 80 columns) and the type of map to produce. The load map options are:

- **s** Lists only load statistics.
- **a** Sorts block map by address. The default.
- **al** Sorts block map by name.
- **b** Restricts block map to objects from *objfiles*.
- **c** Lists common block cross-references.
- **e** Lists entry point cross-references.
- **p** **a** and **al**.
- **f** All the above.

–n Generates a shared text program on the Cray X–MP.

–N Inhibits inclusion of the default libraries in the load.

–o *outfile* Writes the executable program to *outfile*. Defaults to the name specified in an **ABS** directive, or to **a.out**.

–O *keyword* Selects memory allocation order according to *keyword*.

- **cm** All common blocks first.
- **mc** All module local blocks first.
- **tdb** All code first, then all initialized data, then all uninitialized data.
- **ema** Allocates code to maximize usage of extended memory addressing on the Cray X–MP.
- **s** Allocates code to create a shared text program for the Cray X–MP.
- **ss.ema** Allocates code to create a split-segment program to maximize use of EMA on the Cray X–MP.
- **ss.tdb** Allocates code to create a split-segment program, with code first, then initialized data, then uninitialized data.

–s Inhibits the generation of debug symbol tables.

–S *si*[+*se*] Assigns the initial stack size (*si*) and the stack expansion increment (*se*).

–t Trial mode. **segldr** scans all object modules, checks errors, and generates load maps, but produces no executable program.

–u *unames* Enters *unames* as undefined entry names. Useful to force loading of desired routines from a library.

–V Writes **segldr**'s version line to stderr.

–z Specifies an alternate default directives file. The alternate directives must configure the program correctly for execution under UNICOS.

–Z	Inhibits reading of the default directives file **/lib/segdirs/def_seg**, which is required for correct loading under UNICOS. Special purpose programs only.
objfiles	A blank-separated list of file names suffixed with **.o** to be loaded, file names suffixed with **.a** to be processed as libraries, and names of ASCII files containing directives. **–i** is recommended for specifying directives files; **–l**, for libraries.

10.5.2 segldr: Loader Directives

segldr directives provide the ability to restrict the effect of certain **segldr** options to certain portions of the files being loaded, and provide certain capabilities (e.g., segmentation) not available through **segldr** options. **segldr** directives may be input on the command line via the **–D** option, or they may be listed in a file named on the command line. They are separated by newlines or by semicolons. The form of all **segldr** directives is

DIRECTIVE[=*option1*[,*option2*[,...]]]

Of the approximately 60 directives, some of the more commonly used ones follow:

ABS	Names file to receive the executable program.
BIN	Names **.o** and **.a** files to be searched for needed elements.
*	Indicates a comment.
CPUCHECK	Checking of machine characteristics turned **ON** (default) or **OFF**.
DEFDIR	Replaces the default directory search list (**/lib**, then **/usr/lib**).
DEFLIB	Adds libraries to the end of the default list.
DUPENTRY	Specifies the severity level of messages for duplicated entry point errors.
DUPLOAD	Specifies the severity level of messages for common block initialization by more than one module.
ECHO	Printing of directives turned **ON** or **OFF** (default).
INCLUDE	The named directives file is processed immediately.
LIB	Names additional files to be searched for needed entry points.
LIBDIR	Adds names to the beginning of the system's directory search list for libraries and directives files.
MAP	Specifies which load maps are to be produced.
MLEVEL	Specifies the lowest level **segldr** message to be printed (default: **caution**).
MSGLEVEL	Specifies severity level for specific messages.
NODEFLIB	Ignores all default libraries.
NODUPMSG	Suppresses duplicate entry point messages for specific entry points.
NOUSXMSG	Suppresses unsatisfied external messages for specific externals.
PRESET	Specifies a value used to preset uninitialized data areas.
REDEF	Specifies the severity level of messages for redefined common block errors.
SYMBOLS	Generation of debug symbol table turned **ON** (default) or **OFF**.
SYSTEM	Selects the target operating system on which the program will execute.
TRIAL	All object modules are scanned, errors checked, and load maps generated, but no executable code is produced.

USX Specifies the severity level of messages for unsatisfied external errors.

10.5.3 segldr: Environment Variables

The **segldr** command processes the following five environment variables:

SEGLDR Contains one or more strings separated by semicolons where each string is a **segldr** directive or the name of a file containing **segldr** directives.

TMPDIR Specifies the directory that the loader uses for its temporary file. The default directory may be specific to each system.

LPP Specifies the number of lines to print on each page of listing output. Number must be between **15** and **999**, and the default is **57**.

MSG_FORMAT Describes a format specification similar to that of the C library routine printf; this specification can be used to alter **segldr** error message displays.

NLSPATH Specifies a list of alternate directories that the loader should search for its error message catalog. It is used to select alternate catalogs for debugging, or when different versions of **segldr** are operating on the same system. **NLSPATH** is not needed for normal operations.

10.6 Some Nondefault Libraries

DISSPLA 11.0 libraries (Cray X–MP), in **/usr/local/dissplа11/lib**

libdcc.a	(Calcomp Interface)
libdis77.a	(Utility Routines)
libgks.a	(GKS)
libint.a	(Driver Interface)
libpvi.a	(PVI Interface)

DISSPLA 10.0 libraries (Cray–2), in **/usr/local/lib/dissplа.10**

dcclib.a	(Calcomp Interface)
dislib.a	(Utility Routines)
gkslib.a	(GKS)
intlib.a	(Driver Interface)
pvilib.a	(PVI Interface)

Math libraries, in **/usr/local/lib**

libimsl.a	(IMSL library)
libcm.a	(NIST Math Library)

BRL–CAD libraries, in **/usr/brlcad/lib**

libcursor.a	**libnurb.a**	**librt.a**
libfb.a	**liborle.a**	**libsysv.a**
libfft.a	**libpkg.a**	**libtermio.a**
libfont.a	**libplot3.a**	**libwdb.a**
libmalloc.a	**librle.a**	

11. Fortran I/O

This chapter presents an overview of certain features of ANSI Standard Fortran 77 I/O operations as implemented on the ARLSCF Cray computers running under UNICOS release 6.1, and also of certain Cray extensions (not ANSI standard) to Fortran 77 I/O, some of which correspond to ANSI Standard Fortran 90 usage. Extensions to or variations from the 77 standard are so indicated. Because the use of non-standard features diminishes the portability of Fortran code, it is recommended most strongly that non-standard features not be used, unless significant benefits accrue through their use. A more complete discussion, and additional features, can be found in chapters 7, 8, and 9 of the Cray publication, *CF77 Compiling System, Volume 1: Fortran Reference Manual*, SR–3071 5.0, and in the Cray publications, *Segment Loader (SEGLDR) and ld Reference Manual*, SR–0066 6.0, *UNICOS File Formats and Special Files Reference Manual*, SR–2014 6.0, *Volume 1: UNICOS Fortran Library Reference Manual*, SR–2079 6.0, and *UNICOS I/O Technical Note*, SN–3075 6.0.

Throughout this chapter, except at the end, where they are discussed, those entities called *internal files* are entirely ignored. Accordingly, *file* always means *external file*, except at the end of this chapter.

A remark must be made concerning the major software systems pertinent to this chapter. The ARLSCF Cray computers run UNICOS, release 6.1. The Fortran dialect is Cray Fortran release 5.0. At the writing of this chapter, the file system in use is release 5, with installation of the release 6 file system pending. The choice of file system is almost, but not quite, transparent to the Fortran programmer. Such differences are identified as they are encountered in this chapter.

11.1 Files

A *file* is a structured collection of information, created in, maintained in, and removed from external (typically disk) storage by the operating system. UNICOS files are created as *permanent* files; unlike some other operating systems, UNICOS creates no *temporary* files which must explicitly be made permanent lest they disappear.

11.1.1 Formatted/Unformatted

A *formatted* file consists of ASCII characters and can be viewed more or less directly, as on a terminal screen or line printer. It can be manipulated by any of a multitude of UNICOS utilities to accomplish tasks such as horizontal or vertical cut and paste, searching, sorting, counting, and editing. An *unformatted* file consists of binary data and cannot be viewed directly, nor can intuitive tasks like those just mentioned be performed directly thereon.

11.1.2 Sequential/Direct

A *sequential* file can be thought of as information stored on magnetic tape. When the file is positioned at some datum, some other datum can be accessed only by traversing all the intervening data. Data in a *direct* file can be accessed in any order, without traversing intervening data.

11.1.3 Records

Files consist of *records*. A record is the smallest entity which can be manipulated by a Fortran I/O statement. A formatted record is a series of characters, typically a line, with length equal to the number of characters therein. Fortran formatted record maximum lengths are restricted† to a default maximum length of 267 characters on the Cray X–MP and to 1024 characters on the Cray–2. These maximum lengths can be further restricted by the physical characteristics of the specific hardware devices which serve as source or destination of the record. Except for restrictions imposed by hardware devices, the default maximum record lengths may be increased on the Cray–2 by invoking the **WNLLONG** subroutine (see the Cray publication, *Volume 1: UNICOS Fortran Library Reference Manual*, SR–2079 6.0), and on the X–MP by using **segldr** directives. For example, to increase the maximum length of a formatted input record to 384 characters, and of a formatted output record to 522 characters, on the X–MP, use the following directives. Note that the **COMMONS** directives specify values greater by exactly 9 than the desired maximum lengths.

```
SET=$WBUFLN:522
COMMONS=$WFDCOM:531
SET=$RBUFLN:384
COMMONS=$RFDCOM:393
```

Each formatted record ends with a newline character† (octal 012), which is not included in the record's character count.

Unformatted records are not constrained to a maximum length, except by hardware.† Their lengths are measured in words for noncharacter data or bytes for character data. Complex and double precision data require two words per datum. A character datum of length *len* requires *((len–1)/8)+1* words. Other data types require one word per datum.

11.1.4 Multifile Files

UNICOS supports multifile files on the Cray X–MP but not on the Cray–2. A multifile file is referred to by name as a single file, but it contains embedded end-of-file (EOF) records. Such records can be written by the Fortran **ENDFILE** statement and detected by the **READ** statement. Only unformatted files can be multifile files.

11.2 Files and Unit Identifiers

A Fortran program obtains information from, or sends information to, a file by executing one of several data transfer statements (members of the set of I/O statements), the more commonly used ones being:

READ
WRITE
PRINT
BUFFER IN†
BUFFER OUT†
READMS†
WRITEMS†

† Not ANSI standard.

An I/O statement references a file by a *unit identifier*, and the operating system, by a *filename* or some other method. A unit identifier is a nonnegative integer expression, or an "*". The filename follows normal UNIX syntax and can be the file's name, the file's relative path name, or the file's full path name, as appropriate. UNICOS file system release 5 permits filenames up to 14 characters long and path names up to 128 characters long. UNICOS file system release 6 permits filenames up to 256 characters long and full path names up to 1023 characters long. As an example of "some other method," UNICOS can reference its system input, system error, and system output files (commonly known as stdin [standard in], stderr, and stdout) even though these files are *unnamed.*

For an I/O statement to be able to reference a file, the unit identifier and the filename must somehow be associated with each other, connected to each other, or **OPEN**ed. All of these terms, at least in practice on the ARLSCF Crays, are equivalent. Certain portions of the following discussion concerning file opening, connection, and existence may differ somewhat from the ANSI standard, but the discussion is in accord with the realities of Fortran computing on the Cray computers at the ARLSCF.

The most direct method for a Fortran program to open a file is by use of the **OPEN** statement, a simple form of which is, for example

OPEN (8,FILE='myfile')

This particular statement associates unit 8 with file "myfile" in the directory from which the Fortran program is executed. Because all other values which might have been specified within the parentheses have been defaulted, the unit and file are **OPEN**ed for sequential, formatted output, and the file is positioned in such a way that its first† record is ready for access.

Note that nothing has been said about the various UNICOS permissions which obtain for the file directly (owner, group, other) and by virtue of its location in a hierarchy of UNICOS directories. It is the responsibility of the person executing the program to ensure that these permissions are appropriate to the intended use of the file.

11.3 The OPEN Statement

The **OPEN** statement is used to establish or alter a connection between a unit and a file, creating the file if necessary. Its effect is global. Its syntax is as follows. "[]" indicate optional items. Order of the various items is immaterial, except that *u* must be first if **UNIT=** is not present. Values are provided as expressions (constants, symbolic constants, variables, array elements, function references, and combinations thereof) of the appropriate type. If the file already exists, the characteristics specified, explicitly or by default, must be appropriate to the file.

OPEN (	[**UNIT=**]*u*	*u* is the desired unit designator; 0–99.†
	[,**FILE=***fn*]	*fn* is the desired filename; defaults to **fort.***u*;† of type character.
	[,**STATUS=***sta*]	*sta* is **'OLD'** to specify an existing file; requires **FILE=***fn*.
		'NEW' to specify a nonexisting file, which will be created; requires **FILE=***fn*.
		'UNKNOWN' to specify either an existing or a nonexisting file; the default.
		'SCRATCH' to specify an unnamed, previously nonexisting, file which will exist no longer than until program termination; **FILE=***fn* not permitted.

† Not ANSI standard.

[,IOSTAT=*ios*]	*ios* is the name of a type integer storage location into which will be placed the error status of the **OPEN** upon its termination.
[,ERR=*sl*]	*sl* is a statement label to which control will be transferred if an error occurs during the **OPEN**; default is the system's own error handling.
[,FORM=*fm*]	*fm* is **'FORMATTED'** or **'UNFORMATTED'** to specify the type of data transfer for which the connection is made; default is **'FORMATTED'** for **'SEQUENTIAL'** access or **'UNFORMATTED'** for **'DIRECT'** access.
[,ACCESS=*acs*]	*acs* is **'SEQUENTIAL'** for sequential access or **'DIRECT'** for direct access; default is **'SEQUENTIAL'**.
[,RECL=*rl*]	*rl* is the record length in bytes of the records in a **'FORMATTED'** or **'UNFORMATTED'** **'DIRECT'** access file; of type integer.
[,BLANK=*blnk*]	*blnk* is **'NULL'** to ignore blanks in numeric input fields (except that an entirely blank field is interpreted as zero) or **'ZERO'** to treat non-leading blanks in numeric input fields as zeroes; default is **'NULL'**.
[,POSITION=*p*]†	applies only to **'SEQUENTIAL'** files. *p* is **'REWIND'** to initially position the file at its beginning. **'APPEND'** to initially position the file at its end, just before the EOF record. **'ASIS'** to leave an existing file at its most recent position; equivalent to **'REWIND'** if the hardware device does not retain the current position across I/O statements, or if the current position is undefined, which is the case after a **CLOSE**; the default.
[,ACTION=*acn*]†	*acn* is **'READ'** to prevent writing to the file or **'WRITE'** to prevent reading from the file or **'READWRITE'** to prevent neither; defaults to the file permissions.
[,DELIM=*dlm*]†	*dlm* is **'APOSTROPHE'**, **'QUOTE'**, or **'NONE'** to specify the character to be used as a delimiter for type character constants in namelist† or list-directed I/O; default is **'NONE'**.
[,PAD=*pad*]†	*pad* is **'YES'** or **'NO'** to specify whether formatted input is padded on the right with blanks when it is shorter than the format specification.

)

11.4 The CLOSE Statement

The **CLOSE** statement is used to break a connection between a unit and a file, allowing a formerly connected file to be connected to some other unit, a formerly connected unit to be connected to some other file, or the unit and file to be reconnected to each other with the same or different connection characteristics. Its effect is global. Its syntax is as follows. "[]" indicate optional items. Order of the various items is immaterial, except that *u* must be first if **UNIT=** is not present. Values are provided as expressions (constants, symbolic constants, variables, array elements, function references, and combinations thereof) of the appropriate type.

† Not ANSI standard.

CLOSE ([**UNIT=**]*u* *u* is the desired unit designator; 0–102†; 100–102 have no effect.†

[,**STATUS=***sta*] *sta* is **'KEEP'** to specify that the file associated with unit *u* continues in existence after being **CLOSE**d; the default for non-**'SCRATCH'** files; must not be used with **'SCRATCH'** files.

'DELETE' to specify that the file associated with unit *u* ceases to exist after being **CLOSE**d; the default for **'SCRATCH'** files.

[,**IOSTAT=***ios*] *ios* is the name of a type integer storage location into which will be placed the error status of the **CLOSE** upon its termination.

[,**ERR=***sl*] *sl* is a statement label to which control will be transferred if an error occurs during the **CLOSE**; default is the system's own error handling.

)

11.5 Connections

The maximum number of files which can be open simultaneously per process is 60 on the Cray X–MP and 100 on the Cray–2.† Standard input, standard output, and standard error are always open and do not contribute to the count of open files. **cft77** supports units 0–102† and *.

Units 0–102† and * are preopened for formatted sequential I/O as follows, the * connection depending upon whether the * appears in an input or an output statement.

unit	file
*, 5, 100	standard input
*, 6, 101	standard output
0, 102	standard error
other *u*	**fort.***u*

A "preopened" file is one which, without the **OPEN** having been executed, is nevertheless connected to a particular unit. The connection has certain characteristics, just as if the program had executed an appropriate **OPEN** statement. Standard input, standard output, and standard error cannot be connected to units other than those specified because they are *unnamed* files, and so there is no way for a Fortran program to associate them with other units. Unit * is the preferred unit for referencing standard input and standard output because the association is defined within the ANSI standard. Units *, 100, 101, and 102 cannot appear in **OPEN** statements, nor can their connections be altered in any way. The connections for units 0, 5, and 6 can be altered, broken, and reestablished, or even broken and the units then connected to some other file.

An **OPEN** statement whose unit is not currently connected to some user-specified file (by virtue of an earlier **OPEN** statement) and which does not include the **FILE=** specifier, automatically refers to file **fort.***u*, where *u* is the unit number, except that if the unit number is 5, 6, or 0, then the file referred to is standard input, standard output, or standard error, respectively. If necessary, file **fort.***u* is created with size 0 upon execution of the **OPEN**.

An **OPEN** statement whose unit is currently open and which references the file connected to that unit (by default or explicitly with a **FILE=** specifier) can change only the value of the **BLANK=** specifier.

† Not ANSI standard.

An **OPEN** statement whose unit, *u*, is currently open and which references some file other than the one connected to *u* has effects exactly as if it were immediately preceded by the statement

CLOSE (*u*)

A connected file cannot be connected to some other unit without first closing the existing connection.

11.6 Alternatives to OPEN and CLOSE Statements

At program termination, all files still open within that program are **CLOSE**d with default specifier values.

If a unit number *u* other than *, 100, 101, 102 appears in a **READ** or a **WRITE** statement which executes while that unit is not open, then the effect is exactly as if the statement

OPEN (*u*),

were executed immediately before the **READ** or **WRITE** statement.

The UNIX concepts of piping and redirection can be used to permit Fortran programs to access files of various names, of which files the programs have no knowledge at all. For example, if pgm1, pgm2, pgm3, and pgm4 are names of executable files, each of which contains the executable code for a (different) Fortran program which reads from standard input and writes to standard output, then

pgm1 < *inputfile* > *outputfile*

or

cat *inputfile* | **pgm1** > *outputfile*

has the effect that pgm1 reads from file *inputfile* and writes to file *outputfile*, while

(**pgm2** < *inputfile*) | **pgm3** | **pgm4** > *outputfile*

has the effect that pgm2 reads from file *inputfile* and sends its output to pgm3, which reads it and sends its own output to pgm4, which reads it and writes its own output to file *outputfile*. As convenient as this sort of thing may be, it does restrict the number of accessible files to two (or three, if standard error is considered).

The UNIX **ln** (link) command permits any number (up to the maximum; see the preceding section, "Connections") of files to be accessed by a Fortran program, of which files' names the program has no knowledge at all. For example,

ln *myfile* **fort.***u*

permits a Fortran program to access the file *myfile* while opening only the file **fort.***u*, with or without an **OPEN** statement.

We recommend strongly that, with the exception of referring to standard input and standard output as unit * and using the techniques of piping and redirection, none of these techniques be used. Units ought to be explicitly **OPEN**ed and **CLOSE**d with appropriate specifiers and meaningful filenames, such filenames being read as character data by the program, interactively or from a file.

11.7 Data Transfer

11.7.1 READ, WRITE, and PRINT Statements

The ARLSCF Crays provide the ANSI standard data transfer statements **READ**, **WRITE**, and **PRINT**, which can communicate with sequential and direct access, formatted and unformatted files. Their syntax is

READ (clist) iolist
WRITE (clist) iolist
PRINT f, iolist

where *iolist* is the list of entities whose values are to be transferred, *clist* is a list of control specifiers, and *f* is as described under the **FMT** specifier, following.

In **READ** statements, the entities in the I/O list must be names of storage locations which are to receive data, and implied **DO** loops specifying such locations. In **WRITE** and **PRINT** statements, the entities in the I/O list are expressions and implied **DO** loops specifying such expressions, except that any functions specified therein must not themselves invoke data transfer statements, not even to transmit error messages.

Description of *clist* follows. "[]" indicate optional items. Order of the various items is immaterial, except as indicated. Values are provided as expressions (constants, symbolic constants, variables, array elements, function references, and combinations thereof) of the appropriate type.

[**UNIT=**]*u*	*u* is the desired unit designator; 0–102†, *; if **UNIT=** is omitted, *u* must be the first item in the list.
[,[**FMT=**]*f*]	*f* is a statement label referencing a **FORMAT** statement, a character expression whose value is a format specification, or an * denoting list-directed I/O; if **UNIT=** appears, then *f* cannot be specified without **FMT=**; if neither **UNIT=** nor **FMT=** appear, then *f*, if it appears, must be the second item in the list; the format must not specify record lengths greater than the current maximum; presence of *f* identifies the statement as one performing formatted I/O; absence identifies the statement as one performing unformatted I/O.
[,**END=***sn*]	*sn* is a statement label to which control is transferred if an EOF is detected during a **READ**; if not used, a fatal error occurs at EOF; used only with sequential **READ**s.
[,**REC=***rn*]	*rn* is an integer > 0 indicating the record where a direct access I/O operation is to begin; prohibited with sequential I/O.
[,**ERR=***s*]	*s* is a statement label to which control is transferred if a recoverable error occurs during the I/O operation; default is the system's own error handling.
[,**IOSTAT=***ios*]	*ios* is the name of a type integer storage location into which will be placed the error status of the I/O operation upon its termination.

11.7.2 'FORMATTED' and 'UNFORMATTED' I/O

Formatted I/O reads and writes data in the form of ASCII† characters. The data files are suitable for printing and viewing on terminals with a minimum of processing, and, generally speaking, are portable across computing systems. The *format* provides the rules according to which the values of the entities in

† Not ANSI standard.

the iolist are converted between the ASCII† external representation and the binary internal representation. The types of those entities must agree with the types implicit in the various format edit descriptors. Typically, a formatted record may be read correctly using several different formats.

Unformatted I/O reads and writes data in its internal binary representation, without conversion. Typically, the data files are unsuitable for any purpose other than being read on a computing system more or less similar to the one on which they were produced, by a **READ** statement similar to the **WRITE** statement which produced them. In particular,

- The iolist of the **READ** statement must not attempt to read more items than were in the iolist of the **WRITE** statement which wrote to the file.
- The types of the items in the iolist of the **READ** statement must match, item by item, those of the items in the iolist of the **WRITE** statement which wrote to the file.

Because there is no conversion of data, unformatted **READ**s and **WRITE**s are considerably faster than formatted **READ**s and **WRITE**s.

11.7.3 List-Directed I/O

List-directed I/O, denoted by an * in the format position of the control list, is a special case of sequential, formatted I/O.

The effect of list-directed input is as if the Fortran program scanned the input record and combined the manner in which the record is laid out with the program's own knowledge of the type of each entity in the *iolist* to create a format specification, that specification then being used to **READ** the record. Blank, comma, slash, asterisk, and end-of-record serve as data item separators and convey certain additional information to the **READ** statement.

A list-directed output statement simply outputs the values of the entities in the *iolist* according to their type, separating them with blanks and commas in a system-dependent manner.

11.7.4 'SEQUENTIAL' and 'DIRECT' I/O

A good conceptual model for **'SEQUENTIAL'** files is a file stored on magnetic tape. On such a file, the various records are written in sequence and can be accessed only in that sequence; to access the $n^{\underline{th}}$ record, the $(n-1)^{\underline{th}}$ record must first be accommodated.

The characteristic of a **'SEQUENTIAL'** file, that intervening records must be traversed, can result in significant inefficiencies when only certain noncontiguous records are to be read, or when records are to be read out of the sequence in which they were written. Another characteristic of Fortran **'SEQUENTIAL'** files, which gives evidence of their tape ancestry, is that, once any embedded record is modified (i.e., rewritten), the portion of the file beyond that record is unusable. Therefore, to modify a certain embedded record, that record and the portion of the file from that record to the end of the file must be rewritten. (Appending records presents no such inconvenience.)

'DIRECT' files eliminate the cited problems by permitting records to be accessed in any sequence. Each record is assigned an ordinal, which serves as an index permitting such access. The penalty for such convenience is that all records in such a file must be of the same length. (There are nonstandard techniques for writing variable length record files with nonsequential characteristics.)

More correctly stated, it is not the file itself which is **'SEQUENTIAL'** or **'DIRECT'**, but rather, the technique used to access it. Any file whose records are all of the same length (and which resides on a device compatible with nonsequential access) may be opened for either **'SEQUENTIAL'** or **'DIRECT'** access. When **OPEN**ed for **'SEQUENTIAL'** access, such a file's records are accessed in their positional sequence: the first, then the second, ... When **OPEN**ed for **'DIRECT'** access, such a file's record numbers are the records' ordinals.

Figure 11.1 presents a code fragment which displays the **OPEN**ing of four files, one corresponding to each of the four combinations of **'FORMATTED'**, **'UNFORMATTED'**, **'SEQUENTIAL'**, and **'DIRECT'**; the **WRITE**ing of computational results thereon for 1000 cases; the **READ**ing of selected cases therefrom; and, finally, their **CLOSE**ing.

*****

```
          . . .
      INTEGER SELECT(3)
      DIMENSION ARRAY(5)
      CHARACTER STRING*16
          . . .
      DATA N, SELECT /3,1,537,1000/
      OPEN (1,FILE='sequ',ACCESS='SEQUENTIAL',
     +      FORM='UNFORMATTED')
      OPEN (2,FILE='diru',ACCESS='DIRECT',RECL=64,
     +      FORM='UNFORMATTED')
      OPEN (3,FILE='seqf',ACCESS='SEQUENTIAL',
     +      FORM='FORMATTED')
      OPEN (4,FILE='dirf',ACCESS='DIRECT',RECL=118,
     +      FORM='FORMATTED')
      DO 100 I=1,1000
          . . .
         WRITE (1)              VAR,STRING,ARRAY
         WRITE (2,      REC=I) VAR,STRING,ARRAY
         WRITE (3,1000)         VAR,STRING,ARRAY
         WRITE (4,1000,REC=I)   VAR,STRING,ARRAY
  100 CONTINUE
      DO 200 I=1,N
         REWIND 1
         REWIND 3
C        NOTE THAT AN IMPLIED DO ON UNIT 1 WOULD
C        MAKE THE INPUT RECORD TOO LONG
         DO 150 J=1,SELECT(I)
            READ (1) VAR,STRING,ARRAY
  150    CONTINUE
         READ (2,      REC=SELECT(I)) VAR,STRING,ARRAY
         READ (3,1000) (VAR,STRING,ARRAY,J=1,SELECT(I))
         READ (4,1000,REC=SELECT(I)) VAR,STRING,ARRAY
          . . .
  200 CONTINUE
      CLOSE (1)
      CLOSE (2)
      CLOSE (3)
      CLOSE (4)
          . . .
 1000 FORMAT (E17.6,A,5E17.6)
      END
```

Figure 11.1. I/O with the Four Combinations of FORM and ACCESS Values

*****

11.7.5 Carriage Control

Oftentimes, Fortran programs output certain printable characters in column 1 to effect carriage control on line printers configured specifically to process Fortran output. On such printers, column 1 is interpreted as containing a carriage control character and its printing is suppressed. The printers available at the ARLSCF do not support this feature; rather, they follow UNIX conventions to achieve carriage control. The UNICOS utility, **asa**, is available to convert Fortran printable files with column 1 carriage control characters into UNICOS printable files. **asa**'s output is to stdout, with error messages to stderr. Its input is from stdin (thus allowing its use in a pipe) or from files named as arguments. For additional information, see the **asa** manual page (i.e., execute **man asa**).

11.7.6 Newline Suppression

ANSI Standard Fortran 77 causes every sequential, formatted **WRITE** statement to go to the next line as its last act, and provides no means by which this may be suppressed, even though such suppression might be useful, e.g., in the output of prompts during interactive input. As an extension to the standard, Cray Fortran provides the edit descriptor, $, which, if in the logically last position of the output format, suppresses movement to a new line upon completion of the **WRITE**.

11.7.7 NAMELIST I/O†

A Cray extension to the ANSI standard provides for **NAMELIST** input and output, which are documented in the CF77 Reference Manual. It is denoted by the **NAMELIST** *group name* in the format specifier position in the control list. This technique permits data to be input in any sequence and according to any format which are desired at the time of input. It is, however, tedious because it can require about twice as much data entry effort as the other formatted input techniques. More importantly, its use is likely to render a program nonportable. Figure 11.2 presents an example of the use of **NAMELIST**.

11.7.8 BUFFER IN and BUFFER OUT Statements†

A Cray extension to the ANSI standard provides for **BUFFER IN** and **BUFFER OUT** input and output, which are documented in the CF77 Reference Manual. This technique provides for *asynchronous* unformatted I/O; that is, once the I/O operation has been initiated, the program does not wait for its completion, but immediately continues on. Considerable performance improvement may be achieved when I/O operations constitute a significant portion of a program's activity and there is sufficient other work to be accomplished while I/O continues, more or less independently of program execution. Use of **BUFFER IN** and **BUFFER OUT** is likely to render a program nonportable.

One **BUFFER IN/BUFFER OUT** operation transfers data to/from a single array or a single common block, in multiples of 512 words. On the Cray–2, the files being accessed must be of the *pure data* structure (see the following section, "File Structures"). The **UNIT** and **LENGTH** functions are available to delay the execution sequence as desired until specific **BUFFER IN** and **BUFFER OUT** operations are completed, and to return status and number of words transferred information. Subsequent **BUFFER IN** and **BUFFER OUT** operations on a given file or unit are not initiated until preceding operations on those units are completed.

† Not ANSI standard.

*****

program:

```
      PROGRAM DEMO
      NAMELIST /IN/ FL,W,H,DEN
      NAMELIST /OUT/ WGT,FL,W,H
      DATA FL,W,H,DEN /4*1./
   10 READ (*,IN,END=20)
      WGT=DEN*FL*W*H
      WRITE (*,OUT)
      GOTO 10
   20 STOP
      END
```

input:

```
columns 1,2
↓↓
 $IN    $        use program internal values
 $IN FL = 3., W = 3. $
 $IN DEN = .5 $
```

output:

```
columns 1,2
↓↓
 &OUTPUT WGT=1., FL=1., W=1., H=1., DEN=1., &END
 &OUTPUT WGT=9., FL=3., W=3., H=1., DEN=1., &END
 &OUTPUT WGT=4.5, FL=3., W=3., H=1., DEN=0.5, &END
```

Figure 11.2. NAMELIST I/O

*****

Figure 11.3 presents an example of the use of **BUFFER IN** and **BUFFER OUT**. The first **BUFFER IN** initiates a transfer of 1024 words from file *in* to array *A*. Execution continues without waiting for completion until the second **BUFFER IN** is reached, at which time there is a delay until the first one completes (because both reference the same unit). This **BUFFER IN** transfers 512 words from file *in* to the specified portion of array *C*. Immediately upon initiation of the second **BUFFER IN**, the **BUFFER OUT** is initiated, and execution continues past the **BUFFER OUT** while input and output are effected.

*****

```
      . . .
DIMENSION A(1024), C(10000)
      . . .
OPEN (32,FILE='in',FORM='UNFORMATTED')
OPEN (22,FILE='out',FORM='UNFORMATTED')
      . . .
BUFFER IN (32,0) (A(1),A(1024))
      . . .
BUFFER IN (32,0) (C(319),C(830))
BUFFER OUT (22,0) (A(1),A(1024))
      . . .
```

Figure 11.3. BUFFER IN and BUFFER OUT I/O

*****

11.7.9 READMS and WRITEMS Statements†

READMS and **WRITEMS** perform I/O similarly to **READ** and **WRITE** with the **'DIRECT'** specifier, except that records no longer need be all of the same length. To use these and other MS subroutines, a program defines an array in storage in which is maintained an index of file positions versus record indices. Actual maintenance of the array is performed by the MS software package. The record keys can be numeric or alphanumeric, and data transfer can be synchronous or asynchronous. Detailed information is available in the Cray publication, *UNICOS I/O Technical Note*, SN–3075 6.0.

11.8 Improving I/O Performance

The following techniques can be used to achieve greater I/O efficiency:

- Using an implied **DO** within an I/O statement is faster than placing the I/O statement within a **DO** loop.
- Specifying an entire array by name is faster than specifying it element by element using an implied **DO** (by a factor of about 4 when transferring a 1000 element array).
- Unformatted I/O is considerably faster than formatted I/O (by a factor of about 250 on the Cray X–MP). The cost is that the output file is almost certain to be unusable on a dissimilar computing system.
- **BUFFER IN**† and **BUFFER OUT**† increase performance (at the cost of rendering a program nonstandard) by allowing other processing to occur during transfers. Even greater transfer rates can be obtained by using ***pure data*** files (see "File Structures," following) with these statements.

† Not ANSI standard.

11.9 Positioning the File

The ANSI standard leaves undefined the position of a sequential file in many situations. The following describes the situation for sequential files on the ARLSCF Crays when the **POSITION=**† specifier is not used in the **OPEN** statement. The **OPEN**ing of the file may have been accomplished by any means.

- When a sequential file is first opened or created, it is positioned at its beginning.†
- A **REWIND** positions a sequential file at its beginning.
- A **READ**, **WRITE**, or **PRINT** statement positions a sequential file just after the record just transmitted, ready to access the next record.
- A **READ** statement which detects an EOF condition does one of two things:
  - If the **END=** specifier was used, control is transferred to the statement indicated, and the file is positioned after the last data record, that is, correctly positioned to append additional data.
  - If the **END=** specifier was not used, a fatal error occurs and the program terminates.

11.10 File Structures†

Apart from the characteristics **'SEQUENTIAL'**, **'DIRECT'**, **'FORMATTED'**, and **'UNFORMATTED'**, ANSI Standard Fortran leaves undefined the scheme according to which data are stored in files. CF77 as implemented on the ARLSCF Crays uses the following UNICOS file structures:

text — This file structure is used for all formatted I/O, both sequential and direct. It consists of a sequence of 8-bit ASCII characters. Each record is terminated with the ASCII newline character (octal 012), which is not included in the record's character count. No record can be longer than the current maximum record length (see the preceding section, "Records"). A UNICOS sequential text file corresponds to the normal UNIX ASCII file.

blocked — This file structure is used by default for sequential, unformatted I/O. Each block contains 512 words, and each record begins on a word boundary. Blocks and records are delimited by control words. When such a file is created by a Fortran program, it is not easily readable except by a Fortran program using a **READ** statement similar to the **WRITE** statement which wrote to the file, and executing on a system similar to the one on which the file was created.

pure data — This file structure is used by default for direct, unformatted I/O. It can be used for other I/O operations. This type of file maximizes transfer rates. There are neither record nor block boundaries. When such a file is created by a Fortran program, it is not easily readable except by a Fortran program using a **READ** statement similar to the **WRITE** statement which wrote to the file, and executing on a system similar to the one on which the file was created.

† Not ANSI standard.

11.11 Internal Files

An internal file is not a file at all, but rather is a type character array, array element, or variable, or some substring thereof. Such entities are called internal files when used with **READ** and **WRITE** statements to perform type conversion, in a manner analogous to the use of (external) files with **READ** and **WRITE** statements to perform type conversion and data transfer between the interior and exterior of a Fortran program.

READ and **WRITE** statements targeting internal files can perform only sequential, formatted I/O which is not list-directed. Such **READ** and **WRITE** statements are so identified by replacing the unit identifier, **[UNIT=]*u*, with the internal file identifier, [UNIT=]*internal-file-name***. The internal file name is simply the corresponding array, array element, variable, or substring name.

A **READ** statement targeting an internal file obtains character data from the storage locations which constitute the internal file, performs type conversion as specified by the format, and places the results in locations of the proper type specified by the iolist. A **WRITE** statement targeting an internal file obtains values from locations specified by the iolist, performs type conversion as specified by the format, and places the results in the storage locations which constitute the internal file. For example, to convert the integer value in variable IVAL to a type character value, a Fortran program might contain

```
      •••
CHARACTER STRING*10
      •••
WRITE (STRING,'(I10)') IVAL
      •••
```

The result of this operation is that STRING contains a right-adjusted string, 10 characters long, left-filled with blanks, representing as printable characters the integer whose value is in IVAL. Formerly (prior to the 1977 Fortran standard), such conversions, involving Hollerith or character types, were performed using a variety of nonstandard techniques, or by brute force programming dependent upon the specific binary representation of the alphanumeric data.

If the internal file is a character variable or array element, or substring thereof, it consists of a single record whose length is the length of the character variable, array element, or substring. If the internal file is a character array, it consists of a sequence of records, one per array element, each record's length being the length of an array element. **READ**ing or **WRITE**ing beyond the end of an internal file has the same effect as **READ**ing or **WRITE**ing beyond EOF in an external file, unless the internal file is an assumed size array, in which case no detection is performed. An internal file is always positioned at the beginning of its first record, except during a **READ** or a **WRITE**. An internal file becomes defined by **WRITE**ing thereto, by execution of a **READ** statement which specifies the corresponding character storage locations in its iolist, or by assignment of character data to the corresponding character storage locations. Internal files may not appear as targets of I/O statements other than **READ** and **WRITE**.

12. Fortran Code Conversion

Oftentimes, programs developed and used on a particular machine under a particular operating system with a particular compiler will not produce the same results, or even run to completion, when even one of those three environments is changed. This chapter provides assistance to the programmer/user of a Fortran program who is faced with such a problem. In addition, the information provided should be considered in the development of new programs or parts of programs. Specifically, the Fortran dialect addressed is Cray Fortran release 5.0. Detailed information may be found in the Cray publications, *CF77 Compiling System*,

Volume 1: Fortran Reference Manual, SR–3071 5.0
Volume 2: Compiler Message Manual, SR–3072 5.0
Volume 3: Vectorization Guide, SG–3073 5.0
Volume 4: Parallel Processing Guide, SG–3074 5.0

12.1 ANSI Standard Fortran Programs

Fortran programs which do not violate the 1977 ANSI Fortran Standard (henceforth referred to as "the standard") should be compatible with either of the ARLSCF Cray computers after compilation under **cft77** or, preferably, **cf77**.† Of itself, however, this does not guarantee duplication of results obtained on other machines, nor even execution to the point considered by the programmer/user to be "completion." Nevertheless, we recommend most strongly that all Fortran programs be in conformity with the standard in order to reduce the difficulties encountered when a program is moved to a different machine, operating system, or compiler, and to facilitate the continuing task of program maintenance. If, in addition, the program is structured and modular, then those difficulties will be minimized. The only exceptions to this general principle should be those made to accomplish otherwise impossible tasks, or to achieve significantly greater efficiency.

In any event, any nonstandard portions of the code, and any portions for which the standard admits of more than one interpretation, should be clearly identified with comment lines, and an explanation provided of the programmer's intent.

Some of the more common causes of a standard-conforming Fortran program's failure to reproduce earlier results include:

- The standard does not specify the positioning of a file when it is **OPEN**ed. Thus, a standard-conforming program which does not explicitly position a file after **OPEN**ing it may find the file at its beginning (ready to read the first record) on certain machines and at its end (ready to append an additional record) on others. By default, **cf77** positions files at their beginning when they are **OPEN**ed.
- The standard makes use of the **PROGRAM** statement optional, and permits it to have arguments, which the standard ignores. Such arguments are often used in other environments to provide file connection information. **cf77** ignores all arguments on a **PROGRAM** statement.

† **cft77** is the compiler proper. **cf77** invokes the compiling *system*, which performs considerable optimization in addition to that of **cft77**. The compiling system consists of preprocessors **gpp**, **fpp**, and **fmp**, the compiler itself, **cft77**, and the loader/linker **segldr**. Henceforth, only the term **cf77** will be used to identify the compiler, in accord with the preferred command syntax.

- The standard specifies only that type **DOUBLE PRECISION** provide an approximation to the value of a real number which is of greater precision than that provided by type **REAL**. It does not specify how precise the approximation is to be. Because **REAL** entities use 64-bit words (and **DOUBLE PRECISION** entities, two such words) on the Crays, results are expected to be more precise than on most other machines. In fact, Cray single precision results are likely to be as precise as other machines' double precision results. If a program uses **IF** statements to compare noninteger numeric values without an error band, the more precise Cray approximations may cause the program to follow a different logical path.

12.2 Nonstandard Fortran Programs

The problems just described, which may require significant effort to identify and correct, begin to appear trivial when one considers the seemingly endless difficulties caused by the combination of nonstandard usage with unmodular and unstructured (especially "spaghetti") code. Even when some nonstandard feature used in a program being ported seems to be available under **cf77**, there is no *a priori* assurance that the feature has the same syntax and functionality. The following paragraphs indicate the more common areas in which difficulty may arise when a program, which does not conform to the standard or which incorporates poor programming practice, is ported to the ARLSCF Crays.

12.3 Likely Trouble Areas

12.3.1 Programs from a COS Environment

In addition to the material presented herein, Appendix H of the Cray publication, *CF77 Compiling System, Volume 1: Fortran Reference Manual*, SR–3071 5.0, lists differences between **cft** and **cft77**.

12.3.2 Programs from an IBM Environment

In addition to the material presented herein, pay particular attention to nonstandard names of commonly available, or otherwise standard, intrinsic functions (e.g., ARSIN vs. ASIN, ARCOS vs. ACOS, etc.).

12.3.3 Programs from a VAX Environment

In addition to the material presented herein, pay particular attention to:

- **ACCEPT** and **TYPE** statements vs. **READ** and **PRINT**
- **VIRTUAL** statements vs. **DIMENSION**
- VAX specific system calls
- Nonstandard specifiers in **OPEN**, **CLOSE**, and **INQUIRE** statements
- **CHARACTER** and other type entities in the same **COMMON** block
- Use of * in **EXTERNAL** statements
- Edit descriptors without length specification, except for **A**
- Nonstandard use of the **PARAMETER** statement
- **D** in column 1 to indicate debug statements

12.3.4 Character Set

The standard character set is

A through Z
0 through 9
(blank) = + − * / ( ) , . $ ' :

In addition, Cray Fortran recognizes

a through z
" ! _ (tab) @ (@ is used in names created by **cf77**)

Except in type **CHARACTER** or Hollerith values, the compiler does not distinguish between upper and lower case letters. Thus, for example, the keywords **write** and **WRITE** are the same, as are the variable names **result** and **RESULT**.

12.3.5 Lines

cf77, in accord with the standard, accepts Fortran code in columns 1 through 72. The **−N80** option extends this to columns 1 through 80. Lines may be as long as 96 characters; the portion beyond column 72 or column 80 is ignored by the compiler.

The standard accepts as many as 19 continuation lines, permitting a single statement to span as many as 20 lines. **cf77** accepts as many as 99 continuation lines, permitting a single statement to span as many as 100 lines.

Neither the standard nor **cf77** permit more than one statement or parts of more than one statement on one logical line (first line of statement and all of its continuation lines).

12.3.6 Tabs

As an extension to the standard, **cf77** recognizes the tab character within source code, with the following interpretation:

- If a tab character is in column 1, the next character determines the line's interpretation: a nonzero digit indicates a continuation line; otherwise, the line is a statement's initial line.
- A tab character must not occur to the left of a statement label.
- A tab character within the statement is compiled as a blank.
- Tabs are expanded to varying amounts of white space in the compiler listing output, to produce reasonable indentations and other spacing.
- Regardless of how it might be expanded, a tab is counted as contributing one column to the length of a line of source code.

12.3.7 Comments

The standard specifies that any entirely blank line, and any line with a **C** or * in column 1, is a comment. **cf77** extends this to include any line with a **c** or **!** in column 1. Furthermore, embedded comments are recognized: whenever an **!** appears outside of a character constant and not in column 6, the portion of the line to its right is comment. (In column 6, an **!** is a continuation character.)

12.3.8 D in Column 1

Certain compilers treat a D in column 1 as a debugging flag. The **cft77** compiler of the **cf77** compiling system will consider this a fatal error. However, the **fpp** preprocessor of the **cf77** compiling system provides a "switch" (**k**, enabled by default) for use with its enable/disable (**–e**/**–d**) options which causes such statements to be treated as comments. This functionality is available when **fpp** is used in its full precompiler mode and also when all but its TIDY functionality is suppressed.

12.3.9 Names

The standard permits names of programs, subprograms, common blocks, arrays, and variables to be up to six characters long. **cf77** permits those names to be as long as 31 characters.

12.3.10 Local Variables: Retention of Value

Some programs, particularly older ones, depend upon the notion that, once a **RETURN** from a subprogram is effected, variables and arrays local to that subprogram retain their values until that subprogram is again invoked, at which time those values are once again available. That notion is not in accord with the standard, nor is it supported by **cf77**.

Some programs, particularly older ones, depend upon the notion that placing the entities just described in **COMMON** within the subprogram is sufficient to cause retention of their values as just described. That notion is not in accord with the standard, nor is it supported by **cf77**.

In accord with the standard, **cf77** causes all entities (whose values are capable of changing) within a subprogram to become undefined upon execution of a **RETURN** from that subprogram except:

- entities specified therein by **SAVE** statements
- entities specified therein in blank **COMMON**
- entities specified therein in a **DATA** statement, which entities have neither been redefined nor become undefined in any invocation of that subprogram
- entities specified therein in a named **COMMON** block and in the same named **COMMON** block in at least one other program unit that references, directly or indirectly, that subprogram (i.e., other program unit that is in the invocation sequence for the current invocation of that subprogram)

12.3.11 Recursion

As an extension to the standard, **cf77** permits a subprogram to reference itself if its **FUNCTION** or **SUBROUTINE** statement begins with the keyword **RECURSIVE** and it is compiled in stack mode, or if it is compiled with the **cf77** option **–Wf"–o recursive"**.

In accord with the standard, no function whose name appears in the I/O list of an I/O statement may itself invoke I/O statements, not even to issue an error message.

12.3.12 I/O

12.3.12.1 External Files

Connections

Connection and assignment of files and file names by means of arguments on the **PROGRAM** statement (common practice in a CDC environment) is ignored. Files should be connected/disconnected (and created as required) by use of the **OPEN** and **CLOSE** statements.

Standard or System Input and Output

Standard or system input and output should be referenced as unit "*" rather than by some system-dependent number. Unit "*" must be neither **OPEN**ed nor **CLOSE**d. **cf77** preconnects standard error, standard input, and standard output to units 0, 5, 6, respectively, and to units 100, 101, 102, respectively. Connections to units 0, 5, 6 may be changed, but not those to units 100, 101, 102.

Preconnected Files

cf77 preconnects standard error, standard input, and standard output to units 0, 5, 6, respectively, and to units 100, 101, 102, respectively. Units *n* are preconnected to files **fort.***n*, where *n* has values 1, 2, 3, 4, 7, ..., 99.

Positioning

The standard does not specify file position at **OPEN**. Whenever a file is **OPEN**ed with other than **STATUS=NEW**, it should be positioned explicitly at the desired position. **REWIND** positions a file just before its first record, ready to **READ** that record. **READ**ing a file to its end (use the **END=** specifier in the **READ** statement) and **BACKSPACE**ing one record positions it just after its last record, ready to append an additional record.

Number of Open Files

The maximum number of files which can be open simultaneously per process is 60 on the Cray X–MP and 100 on the Cray–2 (nonstandard). Standard input, standard output, and standard error are always open and do not contribute to the count of open files.

Multifile Files

These are not supported in the normal course of events, but may be used on the Cray X–MP for **UNFORMATTED** files if certain system-level action is taken (do **man blocked**).

12.3.12.2 Recursion

In accord with the standard, no function whose name appears in the I/O list of an I/O statement may itself invoke I/O statements, not even to issue an error message.

12.3.12.3 List-Directed Output

The standard provides and **cf77** supports list-directed output. However, the specific appearance of the output (number of items per line, amount and positioning of white space) is not defined by the standard, and is unlikely to be the same under **cf77** as it was in the original environment of the program.

12.3.12.4 NAMELIST I/O

As an extension to the standard, **cf77** provides NAMELIST I/O. Its syntax and operation may not be the same as in the original environment of a program using it.

12.3.12.5 Newline Suppression

ANSI Standard Fortran 77 causes every sequential, formatted **WRITE** statement to go to the next line as its last act and provides no means by which this may be suppressed, even though such suppression might be useful, e.g., in the output of prompts during interactive input. As an extension to the standard, Cray Fortran provides the edit descriptor, **$**, which, if in the logically last position of the output format, suppresses movement to a new line upon completion of the **WRITE**.

12.3.13 INCLUDE Files

Although nonstandard, the **INCLUDE** statement is recognized by **cf77**. It may not be continued onto a second line. The **cf77** option **–I** affects its interpretation.

12.3.14 The PROGRAM Statement

Use of the **PROGRAM** statement is optional under the standard and under **cf77**. Its use is recommended for portability and because certain Cray utilities (e.g., FLOWTRACE) require it. **cf77** ignores all arguments on this statement.

12.3.15 Types

cf77 recognizes types:

REAL	standard
REAL*n*	nonstandard *n*=4, 8; same as **REAL** *n*=16; same as **DOUBLE PRECISION**
DOUBLE PRECISION	standard
DOUBLE	nonstandard; same as **DOUBLE PRECISION**
INTEGER	standard
INTEGER*n*	nonstandard *n*=2, 4, 8; same as **INTEGER**
CHARACTER	standard
LOGICAL	standard
COMPLEX	standard
DOUBLE COMPLEX	nonstandard; see the second following section
POINTER	nonstandard
Boolean	nonstandard; no type statement

In addition to the standard statement **IMPLICIT** *type*, the nonstandard **IMPLICIT NONE** is recognized, requiring explicit typing of all variables and arrays.

12.3.15.1 INTEGER

By default, **cf77** uses a 46-bit representation for integers, for a range

$$-2^{45} \leq I \leq 2^{45}, \text{ or approximately } -10^{14} \leq I \leq 10^{14}$$

64-bit integers can be specified with **cf77 −Wf"−i64..." ...**, for a range

$$-2^{63} \leq I \leq 2^{63}, \text{ or approximately } -10^{19} \leq I \leq 10^{19}$$

12.3.15.2 REAL, DOUBLE PRECISION, COMPLEX, DOUBLE COMPLEX

REAL and **DOUBLE PRECISION** values, and each component of a **COMPLEX** value, have magnitudes in the range

$$2^{-8189} \leq R \leq 2^{8190}, \text{ or approximately } .367*10^{-2465} < R < .273*10^{2466}$$

The **−Wf"−dp"** compilation option causes nonstandard type **DOUBLE COMPLEX** to be recognized, and types **DOUBLE COMPLEX** and **DOUBLE PRECISION** to be compiled exactly as if they were **COMPLEX** and **REAL**. This is a particularly useful option because it permits the use of single precision on the Crays, which is generally at least as precise as double precision on other machines, while avoiding the labor of actually changing the code, with its possibly disastrous results were the code to be returned to some other machine. Single precision on the Crays yields about 14 significant digits, and double precision, about 28.

NOTE CAREFULLY that, if the type **DOUBLE COMPLEX** is used, and the **−Wf"−dp"** compilation option is not used, there will be no error message, nor even a warning or ANSI noncompliance message, except about variable names longer than the standard six characters. The result of the compilation will be to treat all entities, except the first one, in the **DOUBLE COMPLEX** statement as **DOUBLE PRECISION**, and to create a new **DOUBLE PRECISION** variable named **COMPLEX***original-first-name*.

12.3.15.3 CHARACTER

Collating Sequence

The standard requires only that A through Z be in the normal sequence, that 0 through 9 be in the normal sequence, that the letters and digits not be intermixed, and that (blank) occur before both A and 0. **cf77** uses the ASCII collating sequence.

String Delimiters

The standard **CHARACTER** string delimiter is "'". **cf77** accepts "'" and "" " as string delimiters. Of course, the opening delimiter must be the same as the closing one.

Character Relational Expressions

Relational expressions involving two character expressions are permitted by the standard and by **cf77**. However, because the collating sequence is not completely specified by the standard, we recommend the standard intrinsic functions **LGE**, **LGT**, **LLE**, and **LLT** be used. These functions always make comparisons according to the ASCII collating sequence, regardless of which sequence is native to a particular environment. Of course, the equality (**.EQ.** and **.NE.**) comparisons present no such problem.

Hollerith

Hollerith data is nonstandard. It is usable (but obsolete) as a form of Boolean. For portability and greatly increased ease of use (insensitivity to the length of machine words and no necessity to count characters), Hollerith information should be replaced with type **CHARACTER** information.

12.3.15.4 Boolean Values

Boolean values are nonstandard. A Boolean constant displays the internal representation of a 64-bit Cray word. There are no Boolean variables or arrays, nor is there a Boolean type statement. No user-specified function can produce a Boolean result, but certain nonstandard intrinsic functions can.

When one operand of a binary arithmetic or relational operator is Boolean, the operation is performed as if the Boolean operand were of the type of the other operand. When both operands are Boolean, the operation is performed as if both operands were of type **INTEGER**.

A Boolean constant can be written as:

octal	1 through 22 octal digits suffixed **B** (e.g., 177B), or 1 through 22 octal digits delimited with apostrophes or quotation marks prefixed with the letter **O** (e.g., O'177'). Stored right-adjusted in memory with leading zeroes. If all 22 digits are used, the leftmost one can be only 0 or 1.
hexadecimal	1 through 16 hexadecimal digits delimited with apostrophes or quotation marks prefixed **X** (e.g., X'1FF'). Stored right-adjusted in memory with leading zeroes. Digits may be signed (e.g., X'−1FF').
Hollerith	1 through 8 characters prefixed ***n*H**, ***n*** being the character count (e.g., 6HABC 12). 1 through 8 characters delimited with apostrophes or quotation marks suffixed **H** (e.g., 'ABC 12'H). Stored left-adjusted in memory with trailing blanks. If **L** or **R** is used instead of **H**, nulls rather than blanks are used for padding, and storage is left- or right-adjusted, respectively.

12.3.16 Type Conversion

Numeric and Boolean (nonstandard) types may be converted among each other implicitly by using assignment statements, explicitly by using the intrinsic type conversion functions, and implicitly by using mixed type expressions. All three techniques conform to the standard, but the last one is bad practice and fraught with hazard.

Conversion from/to character (including Hollerith) information has been accomplished in any number of ways, from brute force code dependent upon the details of machine architecture and character encoding schemes, to the use of nonstandard facilities such as **DECODE** and **ENCODE**. **cf77** supports **DECODE** and **ENCODE**, but these techniques are obsolete, cumbersome, and vary from one machine to another. The standard (and exceptionally convenient) method of converting from and to type **CHARACTER** is to use **READ** and **WRITE** statements in almost exactly the same manner as for I/O from and to external files. See the chapter, "CF77 I/O," for a brief description.

12.3.17 Logical Expressions

The standard does not specify the machine representations for **.TRUE.** and **.FALSE.**. **cf77** stores **.TRUE.** as a word with high order bit set to one and all others zero. **.FALSE.** is stored as a word with all bits set to zero. Because these representations vary among compilers, the Boolean values for **.TRUE.** and **.FALSE.** should not be used, but rather **.TRUE.** and **.FALSE.** themselves.

12.3.18 Relational Expressions

Whenever relational expressions are used with noninteger numeric entities, error bands should be used to avoid obtaining incorrect results due to insignificant variations in the low order bits. This problem is particularly acute when codes are moved from one machine or compiler to another. Variations are due not only to differing word lengths, but also to differing algorithms used for **REAL** and **DOUBLE**

PRECISION arithmetic.

Character relational expressions (except for those testing equality) should be replaced with the standard intrinsic functions **LGE**, **LGT**, **LLE**, and **LLT** (which always use the ANSI collating sequence) so that variations in collating sequences from one machine or compiler to another will not affect the result.

12.3.19 IF Statements

The 2-branch arithmetic **IF** is nonstandard and obsolete, but still usable under **cf77**. Of form

IF (*exp*) *sl1,sl2*

the statement transfers control to statement label 1 if the arithmetic expression has a nonzero value, otherwise to statement label 2.

The indirect logical **IF** is nonstandard and obsolete, but still usable under **cf77**. Of form

IF (*exp*) *sl1,sl2*

the statement transfers control to statement label 1 if the logical expression has value **.TRUE.**, otherwise to statement label 2.

The 3-branch arithmetic **IF** is standard. Of form

IF (*exp*) *sl1,sl2,sl3*

the statement transfers control to statement label 1 if the arithmetic expression has a negative value, to statement label 2 if the arithmetic expression has a zero value, or to statement label 3 if the arithmetic expression has a positive value.

All three of these statements are easily replaced with logical or block **IF** statements, both of which are standard, greatly improve program readability, and permit the compiling system to perform code optimizations.

12.3.20 DO Loops

Some environments cause **DO** loops (whose **DO** statement is executed) to always be executed at least once, even if the values of the **DO** loop controlling expressions otherwise would cause the loop to be bypassed and not executed at all. This is not the case under **cf77**, which adheres to the standard in this respect. The **–Wf"–ej"** compiling option forces **DO** loops to be executed at least once, regardless of the values of the **DO** loop controlling expressions.

Although the standard permits **DO** loop controlling expressions to be **REAL** or even **DOUBLE PRECISION**, this is extremely bad practice because the iteration count becomes dependent upon insignificant variations in the low order bits. This problem is particularly acute when codes are moved from one machine or compiler to another.

As an extension to the standard, **cf77** supports **DO WHILE** loops and **END DO** statements. The syntax and functionality may not be the same as in a program's original environment.

12.3.21 Array Operations

As an extension to the standard, **cf77** permits the use of arrays as primaries in assignment statements. The operations are elemental; that is, if **A**, **B**, and **C** are conformable arrays, then

```
C = A * B
```

means

```
      DO 20 I = 1, NROWS
         DO 10 J = 1, NCOLS
            C(I,J) = A(I,J) * B(I,J)
10       CONTINUE
20    CONTINUE
```

12.3.22 Masking, Shifting, and Bit Manipulation

As an extension to the standard, **cf77** provides functions which perform logical product, sum, difference, equivalence, complement, shift (circular, left, right), merge, and population count operations. In addition, **cf77** provides integer bit manipulation functions in accord with MIL–STD–1753. Pay particular attention to the effects of differing word lengths between machines.

12.4 Useful Utilities

12.4.1 Code Checking

The **cf77** compiling system itself has excellent code checking capabilities. We recommend that, until a program is determined to be error free, at least the following options be used to provide a full suite of compilation and execution diagnostics.

cf77 –F –g –Wf"–ecimnsx –ooff –m0 –Rabc" *files*.**f**

12.4.2 TIDYing

Code "tidying" or "beautifying," although entirely cosmetic, can make a remarkable improvement in the legibility and, hence, maintainability, of intricate, lengthy, unstructured programs with statement labels not in numerical order and with **FORMAT** and **DATA** statements scattered throughout.

The **cf77** compiling system includes the **fpp** preprocessor, which can produce vectorization of source code beyond that provided by **cft77**. Because of its extensive parsing and analysis capabilities, **fpp** is more than an adequate processor to beautify source code. The command

fpp –dacdehjlmpsuvy015 –ql –r... –ny... *files*.**f**

disables (**–d**) almost all (see the discussion at the end of this section) functionality of **fpp** except TIDYing, with output to standard out. The **–r** and **–n** options use their own set of switches to enable and disable, respectively, various features of code beautification. There are 21 such switches which provide the user with the ability to select exactly what is to be done. Among the more popular features, with very abbreviated descriptions and their default settings, are:

- **a** places inline comments on preceding line if available inline space becomes insufficient (on)
- **c** ensures space after comma in list (on)
- **e, p** **e** ensures space around =, and **p**, around + and – (on)
- **f** positions FORMAT statements just before the **END** (off)
- **j, t** **j** ensures space around ** and //, and **t**, around * and / (off)
- **k, o, q** **k** ensures space around **.AND.**, **.OR.**, **.EQV.**, **.NEQV.** (off); **o** ensures space around all logical operators (off); **q** is a combination of **k** and **o** which treats **IF** statements differently (on); only one of the three may be on

m modifies spacing rules to permit short, otherwise two-line, statements to fit on one line (on)

n ensures space around nonsubscript parentheses (off)

r generates a block of comments listing externals (off)

s applies **j**, **p**, **t** spacing rules to inside of parentheses (off)

x shorthand specification of a popular style: format relabeling initialized at 900, other statement relabeling initialized at 100, both with an increment of 10, and insertion of comments summarizing externals (off)

y beautifies only optimized blocks and echoes remainder of program (on)

These and other features can be selected and modified by the use of directives within the source files, applying to portions or to the entireties of the files. The **SWITCH** directive is the only way to change **FORMAT** and other statement label initial values and increments, indentations for different classes of statements, placement of statement labels, placement of comments, and the continuation line character. For example, the directives

```
CFPP$ SWITCH,FORMAT=900:10,RENUMB=100:10,LABELS=5:R,CONCHR=+
CFPP$ SWITCH,INDDO=3,INDIF=3,INDCN=3,LSTCOL=31
```

renumber **FORMAT** and other statement labels to starting values of 900 and 100, respectively, with increments of 10; right-adjust statement labels into column 5; specify "+" to be the continuation line character; specify that **DO** blocks and **IF** blocks be indented three columns for each nesting level; specify that continuation lines be indented three columns; and specify that column 31 be the last column available for indentation, i.e., that no statement begin beyond column 31. Except for the continuation line character, these illustrative values are the TIDY defaults.

Detailed instructions for the use of the TIDY function of **fpp** can be found in the Cray publication, *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0, pages 33 ff and 293 ff.

As stated previously, it is not quite possible to disable all but the TIDY functionality of **fpp**. One characteristic of **fpp** which cannot be disabled and which is highly visible to the user desiring ANSI Standard Fortran 77 output is the suffixion of certain Cray Fortran **INTRINSIC** subprogram names with an @ symbol. That this characteristic cannot be disabled is not addressed in the level 5.0 Cray documentation. This characteristic causes no difficulty at all, and, in fact, is a desired result, if the resulting code is then to be compiled by **cft77**, with or without intervening **fmp** processing. However, if the output is intended to be Fortran 77 source code with no greater degree of nonstandardness than the input, then the presence of the @ symbols is a corruption of the code.

There are several corrective strategies. One of the simplest is based on the realization that the @ symbols will almost certainly be sparse. In this case, it is practical to execute **grep –n @** on even large source files to identify and examine those lines containing @ symbols. With or without the **grep**, appropriate editors can be used to find and remove the offending @ symbols.

For those cases where the source code is so lengthy or so riddled with @ symbols that manual correction is undesireable, the **fpp** output can be piped through the filter of Figure 12.1 to remove exactly those @ symbols inserted by **fpp**.

12.4.3 fsplit

fsplit [**–s**] [**files**] is a UNICOS utility which splits up one or more files containing Fortran program elements into a number of other files, each one containing one program element. Each main **PROGRAM**, **BLOCKDATA**, **FUNCTION**, and **SUBROUTINE** constitutes one element. This functionality can make less onerous the conversion of larger programs and the casting of such programs into a makefile structure. The **–s** option strips trailing blanks from lines.

*****

```
gawk '
BEGIN { split("MOD@     ABS@     IABS@    INT@     REAL@"     \
              "MAX@     MAX0@    MIN@     MIN0@    AND@"      \
              "IAND@    OR@      IOR@     ISHFT@   SHIFTL@"   \
              "SHIFTR@  CVMGM@   CVMGN@   CVMGP@   CVMGT@"    \
              "CVMGZ@"                                       \
              "mod@     abs@     iabs@    int@     real@"     \
              "max@     max0@    min@     min0@    and@"      \
              "iand@    or@      ior@     ishft@   shiftl@"   \
              "shiftr@  cvmgm@   cvmgn@   cvmgp@   cvmgt@"    \
              "cvmgz@",                                      mod)
      n = split("MOD     ABS      IABS     INT      REAL"     \
              "MAX      MAX0     MIN      MIN0     AND"      \
              "IAND     OR       IOR      ISHFT    SHIFTL"   \
              "SHIFTR   CVMGM    CVMGN    CVMGP    CVMGT"    \
              "CVMGZ"                                        \
              "mod      abs      iabs     int      real"     \
              "max      max0     min      min0     and"      \
              "iand     or       ior      ishft    shiftl"   \
              "shiftr   cvmgm    cvmgn    cvmgp    cvmgt"    \
              "cvmgz",                                       orig)
    }

    { for (i = 1; i <= n; i++)
          gsub(mod[i],orig[i])
          print
    } '
```

Figure 12.1. Filter for Removing Suffixed @ Symbols in **fpp** Output

*****

12.4.4 flint

flint (Fortran–lint) is third-party software which performs a global analysis of Fortran source code and reports a wide range of actual and potential problems and instances of poor programming practice. There are many options to govern its specific behavior. **flint** with neither options nor file names produces a help screen. **flint –demo** executes a small demonstration case. **man flint** yields the man page. A hard copy manual which provides the same information as the man page, but in greater detail, is available.

13. Interlanguage Communication

The considerations expressed in this chapter apply in general to programs whose source codes incorporate more than one language, but the specific techniques pertain to the ARLSCF Cray computers. By the very nature of the topic, much of the material is not within any standard, but certain parts could easily be restricted thereto, e.g., Fortran names could be restricted to being no longer than six characters (ANSI Standard Fortran 77).

For convenience, "program" refers to any portion of a program: the entire program, a main program, subroutines, functions, etc.

Additional information may be found in the Cray publications, *UNICOS Fortran Library Reference Manual*, SR–2079 6.0 (Sections 1 and 8, and Appendix A), *UNICOS Standard C Library Reference Manual*, SR–2080 6.0, (Interlanguage Communications), *UNICOS Math and Scientific Library Reference Manual*, SR–2081 6.0, *Cray Standard C Programmer's Reference Manual*, SR–2074 3.0, *CF77 Compiling System, Volume 1: Fortran Reference Manual*, SR–3071 5.0, and *Interlanguage Programming Conventions Technical Note*, SN–3009.

13.1 Fortran and C

This section presumes CF77, version 5, and the Cray Standard C Compiler, version 3. Earlier compilers may be more restrictive.

In general, the burden of interlanguage communication rests on the C portions of the program; the Fortran portions are written as if they were communicating with other Fortran programs.

13.1.1 Obscure Restriction

There is an ANSI Standard Fortran 77 restriction that dummy arguments, or dummy arguments and entities in **COMMON**, whose storage is made to overlap by virtue of their association with particular actual arguments, must not have their values changed within the referenced subprogram. For example, calling

```
SUBROUTINE SUB(A,B)
```

with

```
CALL SUB(C,C)
```

or calling

```
SUBROUTINE SUB(A)
COMMON /XXX/ B
```

with

```
COMMON /xxx/ C
...
CALL SUB(C)
```

causes overlap between A and B. In either case, the values of A and B must not be changed within the subroutine.

This restriction is extended in Cray Fortran 77 as follows. If the storage of a pointer dereference overlaps the storage of another variable, dummy argument, or pointer dereference, then neither may be modified within that Fortran program.

This restriction does not apply to programs written entirely in ANSI Standard C. The ANSI C standard specifies the exact sequence in which operations are to be performed; therefore, there is no ambiguity concerning which value will be resident in a memory location after the execution of C code equivalent to the Fortran example just presented.

The restriction does apply to Fortran programs invoked by C programs, but not vice versa.

Optimization in general, and in particular each of Cray scalar optimization, vectorization, and parallelization, abrogate the C standard's requirements concerning order of operations (except, of course, that the computation still must be algebraically correct, within precision limits), and so, if there is optimization, the Fortran-like restriction is asserted even in otherwise standard C programs.

Finally, there is a known bug in the Cray Standard C compiler (release 3.0.4.8) which bears upon this topic. For example, the ANSI Standard C program

```
#include <stdio.h>
int com = 1;
sub(int *a){
        com = 2;
        com = *a;
}
main(){
        sub(&com);
        printf("%d\n", com);
        return 0;
}
```

even when compiled with the **–h stdc** option, prints **1**, whereas it should print **2**. A workaround is to program in Standard C as if the Fortran restriction applied. The bug is scheduled for correction in a future release (**cc –V** reports the currently installed version of the C compiler).

13.1.2 Invoking Subprograms

13.1.2.1 Names

Acceptable Fortran names are a subset of acceptable C names. Fortran names are restricted to 31 characters consisting of letters, digits, and the underscore character. The first character must be a letter. Fortran maps all lower case letters into upper case, except for values in character strings. In addition, Fortran will accept certain special characters in global names (functions, subroutines, common blocks), but Fortran programmers should not utilize this feature in creating their own global names. Therefore, the names of C functions invoked from Fortran are restricted to be no longer than 31 characters consisting of upper case letters, digits, and the underscore, and beginning with a letter.

C functions invoked from Fortran are declared either implicitly by use or explicitly with an **EXTERNAL** statement.

The names of Fortran subprograms invoked from C suffer no restrictions beyond normal Fortran requirements. Because Fortran maps lower case letters into upper case, the name by which the program is invoked in C must be upper case, regardless of any lower case which might have been used in writing the Fortran subprogram.

C programs invoking Fortran programs should declare them with the storage class keyword **fortran**, as in

fortran double *function_name* (...)

13.1.2.2 Arguments

Fortran always associates actual and dummy arguments by address. By default, C associates actual and dummy arguments by value, unless they are arrays or executables, in which case they are always associated by address. Therefore, when C and Fortran invoke each other, C programs passing actual arguments other than executables and arrays to Fortran must do so explicitly by address, and C programs receiving actual arguments other than executables and arrays from Fortran must treat the associated dummy arguments as pointers. C and Fortran programs may pass/receive arrays to/from each other just as they normally do (because both use pass by address in this case); however, see the paragraphs concerning names, arrays, and types. Similarly, because both languages pass subprograms by address, C and Fortran programs may pass/receive subprogram names to/from each other just as they normally do; however, see the paragraphs concerning names, arrays, and types. As calling sequences become more intricate, particular attention must be devoted to insuring that arguments always are passed/received according to the proper semantics.

The value returned by a Fortran function is a value, not an address.

Fortran **subroutines** can be invoked from C as if they were Fortran **functions**, with the sole difference that they do not return a value. Therefore, C programs invoking Fortran subroutines should declare them to be of type **void**.

13.1.3 Data Types

13.1.3.1 Correspondence of Types

C	Fortran	bits	
		information	memory
char		8	8
short		24/32	64
int	INTEGER	46/64	64
long	INTEGER	64	64
float	REAL	64	64
double	REAL	64	64
long double	DOUBLE PRECISION	128	128
float complex	COMPLEX	64,64	128
double complex	COMPLEX	64,64	128
	LOGICAL	1	64
	CHARACTER	variable	

13.1.3.2 Fully Compatible

Fortran **INTEGER**, **REAL**, and **COMPLEX** values can be shared directly (by passing as arguments or through **structures** and **COMMON** blocks) with C **int** and **long**, **float** and **double**, and **float complex** and **double complex**, values, respectively.

13.1.3.3 Almost Fully Compatible – Double Precision

Fortran **DOUBLE PRECISION** values can be shared directly (by passing as arguments or through **structures** and **COMMON** blocks) with C **long double** values, as long as the Fortran compilation does not disable **DOUBLE PRECISION**.

13.1.3.4 Almost Fully Compatible – Pointers

Fortran **POINTER** values can be shared directly with those C pointer values which are word addresses.

13.1.3.5 Convertible – Logical Values

When C and Fortran are to communicate logical values to each other, the C program units should incorporate the statement

 # include <fortran.h>

The **fortran.h** header file provides the functions

_btol ( ) returns the **long int** values, Fortran **.FALSE.** and Fortran **.TRUE.**, for the **long int** arguments **0** and nonzero, respectively.

_ltob ( ) returns the **long int** values, **0** and **1**, for the arguments, address of a Fortran **.FALSE.**, and, address of a Fortran **.TRUE.**, respectively.

Figure 13.1 presents a simple C and Fortran program exemplifying the use of these features. A few blank lines have been added to improve readability. Note the **_ltob** invocations in **CFCN** – the arguments are addresses by virtue of the invocation of **CFCN** from Fortran. Had the **_ltob** invocations been in **main**, their arguments, unlike those of **_btol**, would have been prefixed with **&**.

13.1.3.6 Convertible – Character Values

Both Fortran and C support character strings, but with different semantics.

Character strings used in Fortran programs are generally stored in **CHARACTER** variables or arrays thereof rather than in arrays of single characters. **CHARACTER** data is allocated in such a way that the same amount of storage is allocated by each of the following statements, which define a **CHARACTER** variable of length 80 (A), a **CHARACTER** array of 80 elements, each of length 1 (B), and a **CHARACTER** array of 16 elements, each of length 5 (C):

CHARACTER A*80, B(80)*1, C(16)*5

A **CHARACTER** dummy argument can be defined as

CHARACTER*(*)

which means that the length of the argument is not available at compile time and will be passed to the program at execution time.

In general, Fortran **CHARACTER** entities (scalars, individual array elements, and substrings thereof) are not restricted to beginning on word boundaries. Fortran **CHARACTER** entities have an associated length which specifies the number of characters in the entity. A **CHARACTER** dummy argument can be associated with any **CHARACTER** entity. A **CHARACTER** actual argument is a descriptor containing the word address, the bit offset, and the number of characters in the entity.

C does not support a character string type, but, by convention, C **char**acter pointers point to character strings terminated with a zero byte, and several functions in the C library process such character strings. C may allocate character storage which does not begin on word boundaries. A C **char**acter pointer, like a Fortran **CHARACTER** descriptor, contains a character location, but, unlike the descriptor, it does not

*****

```
patton> cat logical1.c

# include <stdio.h>
# include <fortran.h>
# define TRUE 1                 /* the C convention */
# define FALSE 0                /* the C convention */

fortran int FFCN (long int *, long int *);

main ()
{
    long int ctrue, cfalse, ftrue, ffalse;
    int rtnval;

    ctrue  = TRUE;
    cfalse = FALSE;
    ftrue  = _btol(ctrue);
    ffalse = _btol(cfalse);
    printf ("\n ctrue, cfalse =  %d,  %d\n", ctrue, cfalse);
    printf (" ftrue, ffalse = %d,  %d\n", ftrue, ffalse);
    printf (" ********* end main - begin FFCN *********\n");
    rtnval = FFCN (&ftrue, &ffalse);
}

patton> cat logical2.f

      INTEGER FUNCTION FFCN (A,B)
      LOGICAL A, B, C, D
      INTEGER CFCN, RTNVAL

      DATA C,D /.TRUE., .FALSE./

      WRITE (*,*) 'A,      B       =  ', A, B
      WRITE (*,*) 'C,      D       =  ', C, D
      WRITE (*,*) '********* end FFCN - begin CFCN *********'

      RTNVAL = CFCN (C,D)
      FFCN = 0

      RETURN

      END

patton> cat logical3.c

# include <stdio.h>
# include <fortran.h>

CFCN (e,f)
long int *e, *f;
{
    long int ce, cf;
```

```
    printf (" e,      f       = %d,   %d\n", *e, *f);
    ce = _ltob(e);
    cf = _ltob(f);
    printf (" ce,     cf      =  %d,  %d\n", ce, cf);
}

patton> cc -c *.c
logical1.c:
logical3.c:
patton> cf77 -c *.f
logical2.f:
patton> segldr *.o
patton> ./a.out

 ctrue, cfalse =  1,  0
 ftrue, ffalse = -1,  0
 ********* end main - begin FFCN *********
 A,     B      =  T,  F
 C,     D      =  T,  F
 ********* end FFCN - begin CFCN *********
 e,     f      = -1,  0
 ce,    cf     =  1,  0

patton>
```

Figure 13.1. C and Fortran Communication – Logical Values

*****

contain a length; therefore, it cannot be passed to a Fortran program which expects a **CHARACTER** argument.

When C and Fortran are to communicate character values to each other, the C program units should incorporate the statement

 # include <fortran.h>

The **fortran.h** header file provides the type and the functions

_fcd	defined type which matches the Fortran **CHARACTER** descriptor. An object with type **_fcd** can be passed to or received from a Fortran program whose corresponding dummy or actual argument has type **CHARACTER**. **_fcd** is not guaranteed to work in casts.
_cptofcd (*ccp,len*)	returns a Fortran **CHARACTER** descriptor from the C **char**acter pointer *ccp* and **unsigned** byte length *len*. The resulting descriptor points to the same string as *ccp* and can be passed to a Fortran program expecting a **CHARACTER** entity.
_fcdtocp (*fcd*)	returns a C **char**acter pointer from *fcd*. *fcd* is of type **_fcd** and is effectively a Fortran **CHARACTER** descriptor.
_fcdlen (*fcd*)	returns the byte length from *fcd*. *fcd* is of type **_fcd** and is effectively a Fortran **CHARACTER** descriptor.

Figure 13.2 presents a simple C and Fortran program exemplifying the use of these features. A few blank lines have been added to improve readability.

*****

```
patton> cat string1.c

# include <stdio.h>
# include <string.h>
# include <fortran.h>

fortran int FFCN (_fcd,_fcd);

main ()
{
    int clen1, clen2;
    char *cstring1 = "string1", *cstring2 = "string.2";
    int rtnval;

    clen1 = strlen(cstring1);
    clen2 = strlen(cstring2);

    printf ("\n >%s<  %d\n",cstring1,clen1);
    printf (" >%s<  %d\n",cstring2,clen2);
    printf (" ********* end main - begin FFCN *********\n");

    rtnval = FFCN (_cptofcd(cstring1,clen1), _cptofcd(cstring2,clen2
}

patton> cat string2.f

      INTEGER FUNCTION FFCN (A,B)
      CHARACTER*(*) A, B
      CHARACTER*(20) C, D, E
      INTEGER FLEN1, FLEN2, FLEN3, FLEN4, FLEN5
      INTEGER CFCN, RTNVAL

      DATA C,D,E /'string..3', 'string...4', 'string....5'/

      FLEN1 = LEN(A)
      FLEN2 = LEN(B)
      FLEN3 = LEN(C)
      FLEN4 = LEN(D)
      FLEN5 = LEN(E)

      WRITE (*,*) '>', A, '<  ',FLEN1
      WRITE (*,*) '>', B, '<  ',FLEN2
      WRITE (*,*) '>', C, '<  ',FLEN3
      WRITE (*,*) '>', D, '<  ',FLEN4
      WRITE (*,*) '>', E, '<  ',FLEN5
      WRITE (*,*) '********* end FFCN - begin CFCN *********'

      RTNVAL = CFCN(C,D,E)
      FFCN = 0

      RETURN

      END
```

```
patton> cat string3.c

# include <stdio.h>
# include <string.h>
# include <fortran.h>

CFCN (fstring3,fstring4,fstring5)
_fcd fstring3,fstring4,fstring5;
{
    int clen3, clen4, clen5, cclen3, cclen4, cclen5;
    char *cstring3, *cstring4, *cstring5;

    cstring3 = _fcdtocp (fstring3);
    clen3    = _fcdlen  (fstring3);
    cclen3   = strlen   (fstring3);
    cstring4 = _fcdtocp (fstring4);
    cstring4[10] = '\0';
    clen4    = _fcdlen  (fstring4);
    cclen4   = strlen   (fstring4);
    cstring5 = _fcdtocp (fstring5);
    cstring5[11] = '\0';
    clen5    = _fcdlen  (fstring5);
    cclen5   = strlen   (fstring5);

    printf (" >%s<  %d\n",cstring3,clen3,cclen3);
    printf (" >%s<  %d\n",cstring4,clen4,cclen4);
    printf (" >%s<  %d\n",cstring5,clen5,cclen5);
}

patton> cc -c *.c
string1.c:
string3.c:
patton> cf77 -c *.f
string2.f:
patton> segldr *.o
patton> ./a.out

 >string1<  7
 >string.2<  8
 ********* end main - begin FFCN *********
 >string1<  7
 >string.2<  8
 >string..3            <  20
 >string...4           <  20
 >string....5          <  20
 ********* end FFCN - begin CFCN *********
 >string..3            <  20  20
 >string...4<  20  10
 >string....5<  20  11

patton>
```

Figure 13.2. C and Fortran Communication – Character Values

*****

Note the three lines of output from **CFCN**. In each case, **_fcdlen** returns the string length as it is known to the Fortran **CHARACTER** descriptor. The length of the C string, not including the terminating zero byte, is returned by **strlen**. Note also that **printf** output 20 characters for **cstring3**, but 10 and 11, respectively, for **cstring4** and **cstring5**. This is because **FFCN** padded strings **C**, **D**, and **E** out to 20 characters with blanks. The 21^{st} character in each string is a zero byte, and the first zero byte terminates the action of **printf**. (Recall that each of the Fortran strings is simply a local variable and so begins on a word boundary. Each occupies three words with the rightmost 32 bits of the third word left unused. By default, those 32 bits are zero filled.) **printf** output fewer characters for **cstring4** and **cstring5** because zero bytes were explicitly placed at their 11^{th} and 12^{th} (subscript values 10 and 11) character positions, respectively. **cstring1** and **cstring2**, in **main**, are zero byte terminated in the eighth and ninth character positions by virtue of the ", as opposed to ', in the **char** statement.

13.1.4 Arrays

In Fortran, the array subscript lower and upper bounds may be declared explicitly, or only the upper bound be declared explicitly and the lower bound default to 1, in which case the upper bound has the same value as the array size along that dimension. Fortran stores arrays in such a way that, as one progresses sequentially from lower address elements to higher address elements, it is the leftmost subscript which is incremented most rapidly. This is often, and erroneously, identified as "column major" order.

In C, array lower bounds are always zero. C stores arrays in such a way that, as one progresses sequentially from lower address elements to higher address elements, it is the rightmost subscript which is incremented most rapidly. This is often, and erroneously, identified as "row major" order.

When arrays are shared between C and Fortran, by passing them as arguments or through common blocks,

- The subscript bounds may vary from program to program, but the size of the array must remain the same.
- One-dimensional arrays require no special care with respect to the sequence of their elements because there is only one subscript. They can be shared directly, and only the possible differences in the subscript lower bound values need be accounted for.
- Multidimensional arrays, in addition to having the possible differences in the subscript lower bound values accounted for, must have the order of their dimensions reversed in the definitions and the order of their subscripts reversed in the executable statements in one or the other of C and Fortran, and the size of the array along each (transposed) dimension must be the same in both languages, if corresponding elements are to be conveniently addressable in the two languages. Multidimensional arrays defined in C and passed to Cray math and science library programs, in addition to having the possible differences in the subscript lower bound values accounted for, must have the order of their dimensions reversed in the definitions and the order of their subscripts reversed in the executable statements in the C in order to conform to Fortran's storage convention, which is used by the Cray math and science library programs.

  For example, a Fortran array defined as

  INTEGER A (20,0:10)

  could be defined in C code as:

  int b[11][20];

  Then, the same value could be accessed from Fortran as, e.g., **A(2,7)** and from C as **b[7][1]**.
- Arrays of characters and especially of strings can present formidable difficulty and should not be passed between C and Fortran (except, of course, that a Fortran string converts to a C one-dimensional character array, and vice versa). Arrays of the other types already discussed can be passed between C and Fortran as long as each array element occupies the same amount of storage

in each language. In particular, this requires that **DOUBLE PRECISION** not be disabled in the Fortran compilation.

13.1.5 Fortran COMMON and C External Variables

C external variables defined in a **structure** are accessible from Fortran in named **COMMON** blocks, and vice versa. For example,

```
struct
{
    int i;
    long double d;
    float a[10];
} ST;
main()
{ ...
```

and

```
SUBROUTINE FCTN
COMMON /ST/ STI, STD, STA(0:9)
INTEGER STI
DOUBLE PRECISION STD
REAL STA
...
```

both refer to the same defined entities in the expected way. Note that the correspondence is positional within the structure/block, and that, because of the **long double/DOUBLE PRECISION** variable, the Fortran compilation must not disable **DOUBLE PRECISION**.

14. Debugging Tools

14.1 dbx

Available on the ARLSCF computers except for the Crays. Execute **man dbx** on the desired computer for detailed information.

14.2 cdbx

The **cdbx** package is a tool for source level, interactive, symbolic debugging of programs running under UNICOS. It is based on **dbx**, provided with the Fourth Berkeley Software Distribution; **dbx** itself, however, is no longer supported by UNICOS. On the Crays, the command, **dbx**, has become a synonym for **cdbx**. **cdbx** can be used with C, CAL, CFT77, and Pascal programs. Those parts of a program to which one intends to apply **cdbx** must be compiled using special compiler options, which cause the generation of internal tables permitting the tracing of program execution via those symbolic names originally used by the programmer. Clearly, it is of great benefit when those names are mnemonic and when individual names are not used to represent a number of logically different entities.

cdbx may be used in a line-oriented mode or in the X Window System interface mode. The user may choose his interface via an environment variable or explicitly on the command line. In addition, a user may create an initialization file governing the initial actions of **cdbx**. **cdbx** searches for this file upon invocation first as **.cdbxinit** and then as **.dbxinit**, first in the current directory and then in the home directory. In addition, X Window System users may customize that environment with **cdbx** specific entries in their **.Xdefaults** file.

Detailed information may be found in the on-line manual pages and in the Cray publications, *UNICOS CDBX Debugger User's Guide*, SG–2094 6.0, and *UNICOS CDBX Symbolic Debugger Reference Manual*, SR–2091 6.1.

14.2.1 Syntax and Options

cdbx [**–c** *corefile*] [**–e** *efile*] [**–f** *fname*] [**–h** *restartfile*] [**–I** *dir*] [**–l** *lfile*] [**–L**] [**–p** *pid*] [**–r**] [**–s** *symfile*] [**–V**] [*X-Toolkit-options*] [*commandline*]

–c *corefile* Names the *corefile* (default **core**, created upon abnormal termination of the user program) to be examined. Related command: **examine**.

–e *efile* Causes **cdbx** to echo all its inputs to file *efile*. Related command: **echo**.

–f *fname* Names file *fname* to be examined or modified in a nonsymbolic manner. This can be any file; it need not represent a program or a program image. Related command: **fedit**.

–h *restartfile* Names file *restart* which, upon invocation of **cdbx**, is restored to a running process under **cdbx** control. In general, the restrictions which apply to UNICOS restart files (execute **man chkpnt**) apply to *restartfile*. Related commands: **save** and **restore**.

–I *dir* Adds *dir* to the list of directories to be searched by **cdbx** when a file is required. By default **cdbx** searches the current directory and any directory specified in a file's path name on the **cdbx** command line. Related command: **use**.

–l *lfile* Causes **cdbx** to echo all its inputs and outputs to file *lfile*. Related command: **log**.

–L Requests the line-oriented interface to **cdbx**. See the X Toolkit option display.

–p *pid* Specifies the process id of a current process to be placed under **cdbx** control. Related commands: **attach** and **detach**.

–r Begins execution of specified program or restart file under **cdbx** control. In the absence of errors, **cdbx** exits; otherwise, **cdbx** reports the error and issues the **cdbx** prompt.

–s *symfile* Names file *symfile* which contains the symbolic information produced by the language processors and loader. The default is the file named on the command line or **a.out**.

–V Causes **cdbx** to print its version, build, and copyright information.

X Toolkit options: X Window System arguments entered on the **cdbx** command line override values set in **.Xdefaults**.

–bg *color* Selects the window background color.

–bd *color* Selects the window border color.

–bw *number* Selects the window border width in pixels.

–display *display* Specifies the name of the desired X server.

–fg *color* Selects the text and graphics color.

–fn *font* Selects the display text font.

–geometry *geometry* Selects the window's initial size and location.

–iconic Specifies that the application starts in an iconic state, subject to the window manager's interpretation of such a state.

–name *name* Selects the name under which resources for the application are to be found. Useful when using shell aliases to distinguish between invocations of an application; eliminates the need to create links to obtain a different name for the executable.

–rv Simulates reverse video if possible, sometimes by swapping foreground and background colors. Usually for monochromatic displays, and may not be implemented correctly on all systems.

+rv Prevents simulation of reverse video. Useful for overriding related defaults if reverse video does not work correctly.

–title *string* Selects the window title.

–xrm *resourcestring* Selects a resource name and value to override defaults. Can be used to set resources which have no explicit *commandline* options/arguments.

commandline The entire command line which normally invokes the program, including options, arguments, and redirection. There must be white space before each redirection. Must be enclosed in double quotes (") to protect it from the shell, unless it consists of only the executable's name. Related commands: **run** and **rerun**.

When invoked with neither options nor arguments, **cdbx** searches the current directory for file **a.out**, using it for the necessary symbolic information to begin debugging. **cdbx** also searches the current directory for file **core**, using it as the initial image for debugging. If file **core** is not found, **a.out** provides the initial debug image.

Generally, only one image is specified. If multiple images are specified with command line options, **cdbx** selects the first one encountered in the order: image specified by **−p**, **−h**, **−c**, or by **−f** (in that order), the executable named in *commandline* with **−r**, **a.out** from the current directory with **−r**, **core** from the current directory, the executable named in *commandline*, or **a.out** from the current directory.

14.2.2 Commands

After **cdbx** has been initialized, its actions are controlled by a series of commands, of which the more basic ones are summarized below. The commands summarized here provide, among other things, the abilities to

- query the program for the entities of which it is cognizant
- query the program for values and characteristics of entities
- run the program in its entirety, in part, or one source line at a time
- run the program until some specific condition obtains, including any change in the value of a variable
- automatically query the program for values of variables whenever some condition obtains, including any change in the value of a variable
- execute shell commands from within **cdbx**

Note that, in large part, **cdbx** respects the locality of variables and other symbols. Hence, for example, **dump** in a subprogram will not yield the same information as in its invoker.

Most of the commands perform the desired task when they are typed, but **stop**, **trace**, and **when** generate other instructions which are executed later, when the program of interest executes under **cdbx** control. Furthermore, depending upon their usage, these commands may cause repetitive testing during execution, and this can dramatically slow the execution of a program, particularly if it is large or if the machine load is heavy. A more detailed treatment of the entire command suite is available using **cdbx**'s **help** command. Detailed information is available in the references previously cited.

/[*regular_expression*] Search forward in the current source file for *regular_expression*. Defaults to most recent *regular_expression*.

?[*regular_expression*] Search backward in the current source file for *regular_expression*. Defaults to most recent *regular_expression*.

assign *variable* = *expression* Set *variable* to the value of *expression*.

cont Resume execution from the current location.

delete [*args*] Remove active **stop**, **trace**, and **when** commands. *args* is a comma-separated list of designators of such commands. **status** displays all such commands in effect.

do **do** *i* = *m1*, *m2* [,*m3*] {*commands*}
do while *condition* {*commands*}
Execute *commands* n=max(int((m2−m1+m3)/m3),0) times (expressed in Fortran notation), or while *condition* is true. *i* must not be a program variable. The *m*'s must be integers. *m3* defaults to 1. *condition* must return a Boolean result. *commands* is a semicolon separated list.

down [*n*] / **up** [*n*] Move **down**/**up** the calling tree (away from/toward the main program) by *n* calls, default 1.

dump [*name*] Display the names and values of all variables in the named program unit, common block, or module. *name* defaults to all active program units.

echo [*file*] / **unecho** / **log** [*file*] / **unlog** Begin/end echoing or logging the **cdbx** session to *file*, various defaults. The difference is that echoing copies **cdbx** commands, while logging copies **cdbx** commands and output.

edit [*name*] Invoke an editor (selected by the environment) for the current source file (default), or file *name*, or the file containing program unit *name*. Saving the edited file is to disk, not to **cdbx**'s current file.

explain [[**dbg**]*n*] Provide additional information about **cdbx** diagnostic message **dbg***num*. Defaults to the most recent diagnostic.

file *file* Change to source file *file*. Affects the line numbers and labels used by **stop** and **trace**. With no argument, displays the name of the current source file.

func [*name*] Change the current program unit, but not the execution pointer, to the most recent execution of program unit *name*. Thus, after a **func** *name*, **dump** shows values known to *name*, and **step** acts as if no **func** had been issued. Without an argument, displays the name of the current program unit.

gripe Send a message to the system person responsible for **cdbx**. Text is neither echoed nor logged.

help [*list*] Invoke help for a comma-separated *list* of **cdbx** commands and other topics. Without *list*, a list of all topics is displayed.

if *condition* **then** {*commands*} [**else** {*other_commands*}] Execute *commands* if *condition* is true, otherwise execute *other_commands*. *condition* must return a Boolean result. *commands* is a semicolon separated list.

list [*name*|*n1*[,*n2*]] List the next 10 lines of the current source file (default), or 10 lines centered about the first executable line of program unit *name*, or line number *n1*, or line numbers *n1* through *n2*. *n1* and *n2* may be integers, . (current line), or .±*n*. **0** is a synonym for the line number of the last line.

log [*file*] / **unlog** / **echo** [*file*] / **unecho** Begin/end logging or echoing the **cdbx** session to *file*, various defaults. The difference is that logging copies **cdbx** commands and output, while echoing copies **cdbx** commands.

menu / **unmenu** Add/delete an item to/from the X Window command menu. See the references.

next [*n*] / **step** [*n*] Execute the next *n* source lines, default 1. The difference is that, when the next source line is a subprogram (and that subprogram was compiled with debug symbol table information), **step** "steps" through the subprogram, whereas **next** executes the subprogram in its entirety and resumes stepping only after the source language return. Both commands temporarily disable **stop**s.

print *expressions* Display the values of the comma-separated *expressions*.

printf [(]"*format*"[,*expressions*][)] Same as **print** *expressions*, but in C style, with C style formatting, except that %c is not available.

psym *symbol* Display information about *symbol*.

quit End the **cdbx** session.

[**re**]**run** [*args*] Execute the program from the beginning, creating a new program image. Without arguments, **run** uses the preceding execution's argument set, including redirection, whereas **rerun** clears that set. With arguments, **run** and **rerun** function identically, replacing the preceding argument set in its entirety with *args*. *args* is any set of the kinds of items in **cdbx**'s *commandline* argument, not including the executable name.

return [*name*] Provide a quick escape from a subprogram through which one is stepping. Effectively a **cont**, but only through the next source language return, or through the source language return to program unit *name*.

seg List the segments of a segmented program and note which are resident in memory.

sh [*string*] Invoke a shell to execute UNICOS commands. *string* is the command string. *string* defaults to interactive input. Output is not logged.

show *type* Display a list of symbols of the *type* specified in the current program unit, as provided by the symbol table. *type* is **commons**, **labels**, **lines**, **scopes**, or **vars**.

source *file* Cause **cdbx** to read commands from *file*.

status Display the currently active **stop**, **trace**, and **when** commands.

step [*n*] / **next** [*n*] Execute the next *n* source lines, default 1. The difference is that, when the next source line is a subprogram (and that subprogram was compiled with debug symbol table information), **step** steps through the subprogram, whereas **next** executes the subprogram in its entirety and resumes stepping only after the source language return. Both commands temporarily disable **stop**s.

stop

stop
stop if *condition*
stop at *line* [**if** *condition*]
stop in *program_unit* [**if** *condition*]
stop *var* [**if** *condition*]

Set a breakpoint at which execution will be stopped unconditionally, or whenever *condition* is true, or at the source line number or label, or upon entry to the program unit, or whenever the variable's value changes. *condition* must return a Boolean result.

switch *symfile* Equivalent to exiting **cdbx** and then executing **cdbx** *symfile*, where *symfile* is an executable, but without leaving **cdbx**.

trace

trace
trace if *condition*
trace in *process* [**if** *condition*]
trace at *line* [**if** *condition*]
trace *var* [**if** *condition*] [**in** *process*]
trace *expr* **at** *line* [**if** *condition*]
trace *procedure* [**if** *condition*] [**in** *process*]

Display tracing information throughout execution, or whenever *condition* is true, or whenever the process is active, or display the source at the line number or label whenever that line is executed, or display the value of the variable whenever it changes, or display the value of the expression whenever execution reaches the line number or label, or display calling information whenever *procedure* is invoked and return information upon executing the source language return. *condition* must return a Boolean result.

unecho / **echo** [*file*] / **unlog** / **log** [*file*] End/begin echoing or logging the **cdbx** session to file *file*, various defaults. The difference is that echoing copies **cdbx** commands, while logging copies **cdbx** commands and output.

unlog / **log** [*file*] / **unecho** / **echo** [*file*] End/begin logging or echoing the **cdbx** session to file *file*, various defaults. The difference is that logging copies **cdbx** commands and output, while echoing copies **cdbx** commands.

unmenu / **menu** Delete/add an item from/to the X Window command menu. See the references.

up [*n*] / **down** [*n*] Move up/down the calling tree (toward/away from the main program) by *n* calls, default 1.

whatis *symbol* Display the declaration of *symbol*.

when

when *condition* {*commands*}
when in *procedure* {*commands*}
when at *line* {*commands*}

During execution, execute *commands* whenever *condition* is true, or whenever *procedure* is invoked, or whenever execution reaches the line number or label. *condition* must return a Boolean result. *commands* is a semicolon-separated list.

where [*n*] Display the calling tree (traceback) for the most recent *n* calls, default 100.

whereis *name* Display the full qualification of all symbols with *name*.

14.3 debug

debug is a batch-oriented tool for symbolic debugging under UNICOS. It produces a snapshot dump of a running program. The **–D** option is required at compilation time to produce the necessary symbol table. By default, **debug** assumes names **core** for the corefile and **a.out** for the executable and sends its analysis to standard out. Detailed information is available by executing **man debug** and in the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0.

14.4 symdump

symdump is a Cray library routine which produces the same sort of output as does **debug**, that is, a snapshot dump of a running program. It is invoked from within a program with language dependent syntax. For example, it is invoked from within a Fortran program by:

CALL SYMDUMP [(*'options/arguments'* [,*abort_flag*])]

By default, **SYMDUMP** examines the options and reports errors, but does not cause the program to abort. **segldr** option **–l libdb.a** or an equivalent is required.

Detailed information is available in the Cray publication, *Volume 1: UNICOS Fortran Library Reference Manual*, SR–2079 6.0, and by executing **man symdump**.

14.5 adb

The **adb** command invokes the UNIX absolute debugger, which permits a user to examine core files from crashed systems or aborted programs. **adb** is interactive, allows access to global variables, and provides output in several formats. Detailed information for the Cray computers is available in the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0, and for all ARLSCF computers by executing **man adb**.

14.6 lint

lint attempts to detect features of portable C source code which are likely to be bugs, nonportable, or inefficient. **lint** performs type checking more strictly than the compiler, detects unreachable statements, logical expressions whose value is constant, loops not entered at the top, and variables declared but not used. Detailed information for the Cray computers is available in the Cray publication, *UNICOS User Commands Reference Manual*, SR–2011 6.0, and for all ARLSCF computers by executing **man lint**.

14.7 flint

flint (Fortran–lint) is third-party software which performs a global analysis of Fortran source code and reports a wide range of actual and potential problems and instances of poor programming practice. It is, more or less, the Fortran equivalent of **lint**. There are many options to govern its specific behavior. **flint** with neither options nor file names produces a help screen. **flint –demo** runs a small demonstration case.

man flint yields the man page. A hard copy manual which provides the same information as the man page, but in greater detail, is available.

Intentionally Left Blank

15. Optimizing Cray Programs – Preprocessors

This chapter applies specifically to the ARLSCF Cray computers. Run time efficiency is a prime objective of the Cray compiling systems. Even with default options, Cray compilers produce optimized object code. There are various additional options which can produce even greater optimization, including generating object code which makes efficient use of multiple processors. An important consideration in optimization is that the programmer avoid writing source code containing certain features which inhibit optimization.

15.1 Analyzing a Program and Its Performance

This section introduces UNICOS tools which assist a programmer in determining and improving the efficiency of a program. With the exception of **ftref**, which is designed for Fortran programs, these tools are usable with the various languages supported by UNICOS. All the examples are expressed in terms of Fortran and the Bourne shell. Detailed information may be found in the on-line manual pages and in the Cray publication, *UNICOS Performance Utilities Reference Manual*, SR–2040 6.0.

15.1.1 Analyzing Fortran Source Code – ftref

ftref statically analyzes source code to show program structure, common block usage, and information specific to multitasking.

ftref has a number of directives and command line options. The following is a simple example of **ftref**'s use to obtain common block references (**–c full**) and a full call tree (**–t full**). Note the **–c** option on the **cf77** command; the loader is not invoked, because **ftref** performs only a static analysis, using as its input the compiler's listing output.

cf77 –c –Wf"–esx" *program.f*
ftref –c full –t full *program.l* > *program.ref*

Large call trees can be trimmed with options:

–l *n*	Add to restrict **–t full** to the first *n* levels.
–r *subprogram*	Add to restrict **–t full** to the subtree rooted at *subprogram*.
–t part	Substitute to output certain calling information, but not the call tree itself.

15.1.2 Tracing and Timing Programs – Flowtrace

The Flowtrace feature prints calling and timing information about all or selected program units, as monitored during execution. Flowtrace adds considerable overhead, which may distort the results for quickly executing program units.

To use Flowtrace,

- include **–F** on the **cf77** command line to flowtrace the entire program, excluding desired program units with **CDIR$ NOFLOW** directives in those subprograms, or
- do not include **–F** on the **cf77** command line, and place **CDIR$ FLOW** directives in the program units to be flowtraced.

Whenever flowtracing is done, the main program must be flowtraced if the percentages of execution time spent in various program units are to be reliable. This is not necessary to obtain reliable CPU cumulative times with the **SETPLIMQ** subroutine (see the following section, **SETPLIMQ**). Whenever **FLOW/NOFLOW** directives are used, their effect is limited to the program units in which they appear, and the last one within a program unit is effective for the entire program unit. The execution time of subprograms which are not flowtraced will be included in the execution time of their nearest flowtraced invoker. In flowtraced programs, **CALL EXIT** and **CALL ABORT** must be replaced with **STOP**.

Flowtrace outputs its results to file **flow.data**, which is read by **flowview**. **flowview** has a number of options and environment variables, and has an X Window System interface. The command **flowview** by itself is sufficient for viewing **flow.data** within the X Window System, while

flowview –Luch > flow.report

is typical without it. **–L** suppresses the X Window System interface, and is required whenever command line options are used. **u**, **c**, and **h** affect the content and style of the report, and are probably the most frequently used options.

15.1.2.1 The FLOWMARK Subroutine

The **FLOWMARK** subroutine permits Flowtrace to treat sections of code within a program unit as if they themselves were subprograms. **CALL FLOWMARK('*name*'L)** initiates such a section and **CALL FLOWMARK(0)** terminates it. *name* serves as the pseudo-subprogram name, must be no longer than seven printable characters, and must contain no white space.

15.1.2.2 The SETPLIMQ Subroutine

Executing a **CALL SETPLIMQ(*n*)** causes every subsequently executed subprogram invocation and **RETURN**, within program units being flowtraced, to output to standard error a line containing invocation information and cumulative CPU time, up to a total of *n* lines. Within flowtraced portions of a program, the (potentially voluminous) **SETPLIMQ** output may be controlled by careful prediction of the desired *n*, or by bracketing important portions of the flowtraced code with **CALL SETPLIMQ(*n*)** and **CALL SETPLIMQ(0)**. In this usage, the **CALL SETPLIMQ(0)** turns off the output, and *n* need be only "large enough."

Experience suggests that

- The program unit in which the **CALL SETPLIMQ(*n*)** resides need not itself be being flowtraced, unless that program unit is the main program.
- The **CALL SETPLIMQ(*n*)** may occur as many invoking levels as desired above the flowtraced program unit(s), separated from them by as many intervening **NOFLOW** program unit(s) as desired.
- The **CALL SETPLIMQ(*n*)** may occur as many invoking levels deep as desired in the invoking sequence, with some higher levels being flowtraced. Subsequent to execution of the **CALL SETPLIMQ(*n*)**, information will be output for all flowtraced program units in the invoking sequence, even the ones at a higher level than the **CALL SETPLIMQ(*n*)**.
- It is the invoked program unit, rather than its invoker, which must be being flowtraced in order to obtain information about the invoked program unit. Of course, the invoker also may be being flowtraced, and indeed must be if information concerning its invocation is to be obtained.

- If the immediate invoker is not being flowtraced, then the invoked program unit is said to have been invoked by its nearest invoker, with a corresponding stack depth. The invoker of the main program is *SYSTEM.

15.1.3 Timing A Program

"Profiling" provides a measure of the time spent in different parts of a program. The **time** command provides total execution time for a program. Neither of these methods adds any significant overhead, nor do they provide any call tree information.

15.1.3.1 Profiling

When the profiling library, **libprof.a**, is loaded with an executable, the portion of memory in which the executable resides is divided up into pieces of desired size, called "buckets." Bucket size is defined by the user's environmental variable, **PROF_WPB**, and may be as small as one machine word (default: four machine words). At regular intervals during execution, the operating system samples the program's address register and increments by 1 the "hit count" for the bucket encompassing that address. On the Cray X–MP only, the sampling rate may be changed by use of the environmental variable **PROF_RATE**. By default, **PROF_RATE** is set for the fastest rate. Upon normal termination, all the bucket counts are written to file **prof.data**. Executing **prof –x** merges the profiling statistics with symbol information from the original executable, for which the necessary compilation option must have been used. Finally, executing **profview**, which has an X Window System interface, provides a user-oriented display of the profiling statistics. Both **prof** and **profview** have a number of options and several environmental variables which influence their behavior. **prof** may even provide the final profiling output, if desired, but using **profview**, even without the X Window System, is the recommended technique. A simple example of profiling within an X Window System environment is:

cf77 –G –l prof *program.f*
env PROF_WPB=1 ./a.out
prof –x a.out > *program.prof*
profview *program.prof*

The command

profview –LmhDc *program.prof* > *prof.report*

is typical of use without the X Window System. **–L** suppresses the X Window System interface, and is required whenever command line options are used.

15.1.3.2 time

time acts like a shell to run a program while measuring time expended. No special compile or load options are required in creating the executable. The command

time "*commandline*"

reports to standard error the total wall clock time from start to finish of *commandline*, the time spent in actual execution of *commandline*, and the time used by the operating system in the execution of *commandline*.

15.1.4 Monitoring Hardware Performance

The Hardware Performance Monitor hardware (available only on the Cray X–MP) monitors a number of hardware activities. Statistics concerning these activities can be accumulated by **hpm** for an entire

program and by **perftrace** for program units. These statistics provide an indication of how efficiently an executable utilizes the Cray X–MP's hardware capabilities.

15.1.4.1 Hardware Performance by Program – hpm

hpm acts like a shell to run a program while accruing statistics from the Hardware Performance Monitor hardware. No special compile or load options are required in creating the executable. Four separate runs are required to obtain all possible information, one to accrue data from each of four hardware monitor groups, **0** through **3**. For example, interfacing with the X Window System:

```
hpm -r -g0 "commandline" 2> perf.data
hpm -r -g1 "commandline" 2>> perf.data
hpm -r -g2 "commandline" 2>> perf.data
hpm -r -g3 "commandline" 2>> perf.data
perfview
```

The command

perfview –LBuchM *program.prof* > *hpm.report*

is typical of use without the X Window System. **–L** suppresses the X Window System interface, and is required whenever command line options are used.

15.1.4.2 Machine Performance by Program Unit – Perftrace

Perftrace provides the same kinds of information given by **hpm**, but broken down by program unit, at the cost of significant overhead. Perftrace is not usable with multitasked programs. It is used similarly to Flowtrace (preceding); that is, the executable must be compiled with the **–F** option, loaded with the Perftrace library, **libperf.a**, and the **FLOWMARK** subroutine can be used to address parts of the executable rather than its entirety. Unlike **hpm**, a single execution is sufficient to accrue data from all four hardware monitor groups, and the data from several executions may be concatenated into one data file to enhance the accuracy of **perfview**'s results. Note that the Perftrace library and **hpm** are incompatible.

The following example assumes an X Window System interface:

```
cf77 -F -l perf program.f
./a.out > program.perf
perfview
```

The command

perfview –LBuchM *program.perf* > *perftrace.report*

is typical of use without the X Window System. **–L** suppresses the X Window System interface, and is required whenever command line options are used.

15.2 Vectorization – fpp

There are any number of programming techniques which are known to improve program performance in general. A particular technique which applies only to certain machines and compilers is known as "vectorization," and the machines and compilers to which it applies, as vector machines and vectorizing compilers. The Cray machines and compilers at the ARLSCF are vector machines and vectorizing compilers.

Vector processing is a form of parallel processing in which elements of an array are processed in groups, rather than individually. There are no overhead costs associated with interprocess communication because only one CPU is used. Vector processing may be combined with multitasking, wherein multiple CPUs are used.

Vectorization is the process of changing scalar (nonvector) operations to vector operations, and, for many applications, it is the most important optimization feature of a Cray compiling system. For example, on Cray machines, a vector loop typically executes an order of magnitude faster than an equivalent scalar loop. It is important that programmers write source code in such a manner as not to impede the compiling system's ability to recognize vectorizable logic.

The remainder of this section applies specifically to Cray vector processing. It assumes hardware characteristics specific to Cray computers, is expressed entirely in terms of Fortran, and is the briefest of introductions to a topic of considerable breadth.

Of itself, vectorization performed automatically by **cf77 –Zv** or manually by the user does not cause ANSI Standard Fortran 77 source code to become nonstandard, except that certain code sequences may be entirely replaced by calls to highly optimized library subroutines, and except that certain **IF** loops may be replaced by **DO WHILE** loops. The **–Zv** option invokes **fpp**, the Fortran dependence analyzer, prior to invoking **cft77**. Its product is an intermediate file of Fortran source code, restructured to some degree, with certain code sequences replaced by highly optimized library routines, and incorporating compiling system directives (all beginning with a **C** in column 1 so that they appear to other compilers to be comments) which implement vectorization and autotasking. **fpp** always prefers vectorizing to autotasking. The autotasking directives are ignored by **cft77**. For autotasking (and microtasking) directives to be effective, the Fortran multitasking translator, **fmp**, must be invoked before **cft77**. **cf77**'s **–Zp** option invokes **fpp** and then **fmp** before **cft77** to produce autotasked code. All forms of multitasking are discussed in a later section of this chapter.

Detailed information may be found in the on-line manual pages, in the Cray publications, *CF77 Compiling System*,

Volume 1: Fortran Reference Manual, SR–3071 5.0
Volume 2: Compiler Message Manual, SR–3072 5.0
Volume 3: Vectorization Guide, SG–3073 5.0
Volume 4: Parallel Processing Guide, SG–3074 5.0

and in the reference manuals pertinent to other languages.

Considerable detailed discussion of specific techniques applicable to specific programming situations, and many more examples than are presented herein, may be found in the Vectorization Guide. Before embarking upon the effort of detailed interaction with the vectorization process, the programmer should consider whether the potential return is worth the cost.

15.2.1 General Requirements for Vectorization

cf77 –Zv invokes the **fpp** preprocessor and the **cft77** compiler to vectorize automatically a Fortran program.

To be vectorizable, a source code construct must be an array operation or an innermost loop. Array operations are Cray extensions to ANSI Standard Fortran 77 which treat entire arrays with the same syntax as if they were single variables. Such operations are inherently vectorizable. Oftentimes, an innermost loop and a few next-inner loops can be restructured by the programmer, or are restructured automatically by **cf77 –Zv**, so that all the resulting loops are innermost.

Vector and scalar versions of the loop must yield equivalent results. **cf77 –Zv** performs extraordinarily well in this respect. When **cf77 –Zv** errs, it does so by rejecting vectorization candidates which, despite their appearance, really are vectorizable because of the specific nature of the program at hand, because of the nature of the data to be processed, or for other reasons which are beyond its analytical capability. Often, the programmer may be able to restructure the code so that it will vectorize, or he may be able to use compiler directives to force the compiling system to vectorize the program.

The loop must not be "too complex." There is a tradeoff here. Smaller loops are more likely to be vectorized, but larger loops have less overhead than several equivalent smaller loops, and so yield better performance if they do vectorize.

15.2.1.1 Unvectorizable Code

Innermost loops containing any of the following cannot be vectorized:

- I/O statements, even though an implied **DO** list vectorizes.
- **EXTERNAL**s which are not **VFUNCTION**s and which are not expanded inline. **VFUNCTION** is a compiler directive specifying that a vector form of an **EXTERNAL**, written in Cray Assembly Language, is available.
- Certain **INTRINSIC** functions.
- **RETURN**, **STOP**, or **PAUSE** statements. These should be replaced by branches out of the loop, to the statement.
- Arithmetic **IF**s, computed **GOTO**s, and assigned **GOTO**s.
- Backward branches entirely within the loop (other than the one forming the loop).
- A branch into the loop from its exterior (nonstandard in a **DO** loop).
- Any reference to any **CHARACTER** entity.
- Vector dependencies, with some exceptions.

15.2.1.2 Vector Dependencies

A vector dependency is an expression or group of expressions in a loop such that the results of later iterations depend upon results of earlier iterations. Certain vector dependencies can be vectorized. For example, **R = R + A(I)** within a loop is a "reduction" and can be partially vectorized. Partial sums of A are computed with vector code, then added together with scalar code.

Some vector dependencies have a threshold, that is, a number of iterations before dependency occurs. Consider a loop containing **A(I) = A(I–6)**, and starting at 7 with an increment of 1. Such a construct has a threshold of 6. If the threshold is ignored, only the assignments into **A(7)** through **A(12)** are certain to be correct because the assignments into **A(13)** on out use the results of logically earlier assignments which may not yet be physically completed. For vectorization to be safe in such a situation, the vector length (the number of logical operations being physically accomplished at nearly the same time) must not be greater than the threshold. In the present example, performing the assignments in groups of 6 ensures that each succeeding group finds earlier results ready to be used. **cf77 –Zv** selects proper vector lengths (64 maximum) for thresholds greater than 2 and vectorizes. If the threshold cannot be determined at compile time, **cf77 –Zv** vectorizes with additional code to determine the safe vector length at execution time.

If necessary, **cf77 –Zv** will produce both a scalar and a vector version of a loop with an execution time selection test. For example, the following code produces a scalar loop and a vector loop, with the selection test, **if II.LE.0 .OR. II.GE.N**

```
      SUBROUTINE  Q(A,B,II,N)
      REAL  A(*),  B(*)
                  • • •
      DO  10  I=1,N
          A(II+I)  =  A(I)  +  B(I)
10    CONTINUE
                  • • •
```

15.2.1.3 Loops Containing IFs

Of themselves, the following do not prevent vectorization of loops:

- block **IF**s
- vectorizable **IF**s controlling a **GOTO** which transfers out of the loop
- logical **IF**s whose conditional statement either is not a branch or branches forward within the loop

15.2.1.4 Vectorizable Expressions and Statements

In general, vectorizable expressions are arithmetic or logical expressions and statements which are in innermost loops and which consist of combinations of:

Loop invariants	Constants or simple variables referenced but not redefined within the loop. An array element with a loop invariant as a subscript is itself a loop invariant.
Scalar temporaries	Simple variables defined and later referenced within the loop, but not outside the loop.
Loop counters	Integer variables incremented or decremented by an integer constant expression on each pass through the loop. The only operators permitted in the expression are + and −. If the result could alternate in sign, it inhibits vectorization. A loop counter may be incremented or decremented more than once per pass if the effect on the loop is the same as some single assignment.
Vector array references	References to array elements whose subscripts are not loop invariant.

Invocations, with vectorizable expressions as arguments, of **FUNCTION**s with vector versions.

15.2.2 Some Other Considerations

15.2.2.1 Vectors

A *vector* is the basic operand for vectorized operations. It is an array, or a subset of an array, of not more than 64 elements. If more than 64 elements are involved for one operand in the desired computation, they are grouped into additional vectors, each with its own startup and ending overhead in the vectorized operation. Sometimes a sequence of vectorized operations can be chained together to reduce the overhead.

15.2.2.2 The Stride

The *stride* is the interval between memory locations for successive vector elements. Vectorization requires that the stride be constant for all the elements of a vector operand. Not only must the stride be constant, but also it is important that its value not cause memory contention.

15.2.2.3 Memory Contention

Memory on a Cray is arranged in distinct banks. When a memory location in a bank is accessed, that bank becomes unavailable for a certain number of clock periods, the "bank-busy" time. To avoid performance degradation, it is important that successive accesses to the same memory bank not occur within the bank-busy time. The Cray X–MP has 64 banks, and the Cray–2, 128. The **target** command yields the number of banks and the bank-busy time on either machine.

Consider the following code on the Cray X–MP:

```
                    • • •
      REAL  A(256,100),  B(256,100)
                    • • •
      DO  20  I=1,256
          DO  10  J=1,100
              A(I,J)  =  2.*B(I,J)
10        CONTINUE
20    CONTINUE
                    • • •
```

Array elements are accessed in the order A(1,1), A(1,2), A(1,3), ... Because 256 is a multiple of 64, all these accesses (until A(2,1)) are to the same bank, with terrible performance degradation. Were the two **DO** statements reversed, each access would be to a different bank, until A(65,1), at which point the bank access sequence repeats. In the reversed example, there is sufficient time between accesses to the same memory bank that there are no bank-busy delays.

A different solution is simply to increase the arrays' column lengths to 257, without reversing the order of the **DO** statements. The trip count is not changed, only the array dimensions. Memory accesses remain along the rows, but with a stride of 257, each successive access is to the next memory bank (257 modulo 64 = 1). Odd strides guarantee that successive memory accesses are not to the same memory bank.

15.2.3 User Interaction with Vectorization

The **fpp** preprocessor, invoked by **cf77**'s **–Zv** option, produces an intermediate source code file. The intermediate source code file conforms to the ANSI Fortran 77 standard as well as did the original code, except that certain code sequences may be entirely replaced by calls to highly optimized library subroutines, and except that certain **IF** loops may be replaced by **DO WHILE** loops. The differences between the two files consist of restructured code sequences, substituted code, and inserted compiler directives.

The user may select appropriate **cf77 –Zv** options to view the results of and, in some cases, rationale for **fpp**'s processing, to influence or override **fpp**'s processing, to retain **fpp**'s output file, and to influence or override **cft77**'s vectorizing. In addition, both **fpp** and **cft77** understand directives which can provide very specific control over their processing. Partial listings of these options and directives are presented in following sections of this chapter.

A useful technique for the user desiring specific information about the optimization performed, or needing a point of departure for additional or different optimization, is to obtain the **fpp** and **cft77 –em** listing outputs and the **fpp** intermediate source code output. These, of course, may be obtained directly from **cf77**.

15.2.3.1 fpp Options

These options are provided to **fpp** in the normal manner, or to **cf77** in a **–Wd"..."** construct.

–I *subpgm1,subpgm2,...* Lists subprograms which are to be expanded inline.

–l *file* Enables the **fpp** listing and directs it to *file*.

–M *lines* Specifies the maximum number of source *lines* allowed for automatic inline expansion; default 50.

–o *file* Directs the intermediate source code file to *file* instead of to standard output. *file* is suitable for input to **fmp** or **cft77**.

–d *offstring* Disables or
–e *onstring* Enables the following optimization switches:

- **a** Permits associative transformations of code, for example, the vectorizing of a reduction. May cause slightly different results due to the inherent nonassociativity of **REAL** arithmetic. Default: enabled.
- **d** Performs vector dependency checking. Default: enabled.
- **j** Translates certain nested loop constructs, such as element-by-element matrix operations, into calls to highly optimized library routines. Default: enabled.
- **l** Translates **IF** loops into **DO** loops, including nonstandard **DO WHILE** loops. Default: enabled.
- **m** Generates alternative code for potential dependencies. Default: enabled.
- **s** Permits loop splitting to isolate vector dependencies. Default: enabled.
- **t** Uses aggressive criteria for vectorizing nested loops. Default: disabled.
- **v** Enhances the subsequent **cft77** vectorization. Default: enabled.
- **x** Creates the intermediate source code file. If disabled, only the listing file is produced, at some saving of time and disk space. Default: enabled.
- **6** Subprograms meeting certain criteria are expanded inline. The criteria are such that the code produced is always safe. Default: disabled.
- **7** Subprograms meeting certain criteria are expanded inline. The criteria are such that the code produced is rarely unsafe. Default: disabled.

15.2.3.2 fpp Directives

These directives appear in the **fpp** or **cf77** input file and are interpreted by **fpp**. Their prefix of **CFPP$** followed by at least one blank causes them to be comments for non-Cray compilers. Some are of form *directive/opposite_directive*. In such cases, the default is listed first. Some accept a trailing blank and then a scope parameter, which specifies the range of code to which the directive applies. Values of the scope parameter are:

- **R** Directive applies to end of current program unit.
- **L** Directive applies to next **DO** or **DO WHILE** (nonstandard) loop encountered. The default.
- **F** Directive applies to end of current file.
- **I** Directive applies at the current point in the source code.

A subset of the directives is:

ALTCODE[*n*]**/NOALTCODE** Enables/disables generation of alternate code blocks. *n* is an integer indicating a trip count or any other expression to be used in tests for alternate code. Scope: **L/R/F**. Similar to **–em**.

ASSOC/NOASSOC Enables/disables associative transformations. Scope: **L/R/F**. Similar to **–ea**.

NOAUTOEXPAND/AUTOEXPAND Disables/enables automatic inlining. Scope: **L/R/F**.

CHOP_HERE Breaks a loop into two loops at the current point.

DEPCHK/NODEPCHK Enables/disables vector dependency checks. **NODEPCHK** asserts that no vector dependency exists. Scope: **L/R/F**. Similar to **–ed**.

RELATION (*int1 rel int2*) Provides additional information to **fpp** by asserting that a particular relationship exists between an integer variable (*int1*) and another integer entity (*int2*). *rel* is a Fortran relational operator. Scope: **L/R/F**.

SELECT	Selects the next loop in a nest as the one to optimize.
SWITCH	Permits various optimization and listing parameters to be set from within the source file. Certain TIDY parameters (see the TIDY section, following) can be set in no other way.
NOUNROLL/UNROLL[*n*]	Disables/enables loop unrolling. With scope **R** or **F**, *n* is maximum trip count for automatic unrolling (default 3). With scope **L**, *n* is the number of times to unroll the loop, default calculated by **fpp**.

15.2.3.3 cft77 Options

These options are provided to **cft77** in the normal manner, or to **cf77** in a **–Wf"..."** construct.

–I *inlname* Explicit inlining. Every program in file or directory *inlname* is inlined regardless of its calling tree level, as long as it meets certain other requirements. Distinct from automatic inlining (**–o inline**[*n*], *n*=1,2,3), which is applied to all programs whose calling tree level is not greater than *n* and which meet certain other requirements.

–l *listfile* Creates file *listfile* to receive listing output enabled by **–e** options **c**, **g**, **m**, **s**, **x** and by the **LIST** and **CODE** directives. Default: *file*.**l**, from *file*.**f**.

–o *optim* [,*optim*]... Specifies code optimizations to be performed during compilation. A complete list of optimization codes may be found in the chapter, "**cf77** Compiling System."

–d *offstring*
–e *onstring* Disables or Enables various compilation options. A complete list may be found in the chapter, "**cf77** Compiling System." A particularly useful option when vectorization is involved is **–em**. This option specifically marks all loops in the listing output and indicates certain of their characteristics which pertain to their vectorization: whether they are vector, scalar, or unwound loops, and whether they are bottom loaded, unrolled, short, vectorized with a computed maximum safe vector length, vectorized with a short safe vector length, and/or unconditionally vectorized with **CDIR$ IVDEP**.

15.2.3.4 cft77 Directives

The following directives are a small subset of those which the programmer may place in the **cft77** or **cf77** input file and which are interpreted by **cft77**. Certain ones also influence the operation of **fpp**. Their prefix of **CDIR$** followed by at least one blank causes them to be comments for non-Cray compilers. Much of **fpp**'s processing results in the insertion of these same **cft77** directives; **fpp**'s insertions, however, are prefixed **CDIR@**, so that they may be distinguished from user insertions.

IVDEP[**SAFEVL=***n*]	Causes the compiler to ignore vector dependencies when the vector length is at least *n*, default all vector dependencies. Applies to the first **DO** or **DO WHILE** (nonstandard) loop, or array syntax assignment, following the directive and in the same program unit.
NEXTSCALAR	Suppresses vectorization of following **DO** or **DO WHILE** (nonstandard) loop.
RECURRENCE/NORECURRENCE	Toggle on and off the command line enabled vectorization of reduction loops; directives override **–o recurrence** but not **–o norecurrence**. A reduction loop reduces an array to a scalar value by doing a cumulative operation on all of the array's elements; this involves including the result of the previous iteration in the expression of the current iteration.

SHORTLOOP	Declares that a loop has a trip count less than 64, thereby eliminating the need for code to test for completion of that loop. Applies to the first **DO** or **DO WHILE** (nonstandard) loop following and in the same program unit as the directive. Effective only in vectorized loops.
VECTOR/NOVECTOR	Toggles on and off, command line enabled vectorization.
VFUNCTION *list*	Indicates that the listed external functions have vector versions.
VSEARCH/NOVSEARCH	Toggles on and off, command line enabled vectorization of search loops. A search loop is a loop which can be exited by means of an **IF** statement.

15.2.4 Another Capability – TIDYing

Code "tidying" or "beautifying," although entirely cosmetic, can make a remarkable improvement in the legibility and, hence, maintainability, of intricate, lengthy, unstructured programs with statement labels not in numerical order and with **FORMAT** and **DATA** statements scattered throughout.

Because of its extensive parsing and analysis capabilities, **fpp** is more than an adequate processor to beautify source code. The command

fpp –dacdehjlmpsuvy015 –ql –r... –ny... *files*.**f**

disables (**–d**) almost all (see the discussion at the end of this section) functionality of **fpp** except TIDYing, with output to standard out. The **–r** and **–n** options use their own set of switches to enable and disable, respectively, various features of code beautification. There are 21 such switches which provide the user with the ability to select exactly what is to be done. Among the more popular features, with very abbreviated descriptions and their default settings, are:

a	places inline comments on preceding line if available inline space becomes insufficient (on)
c	ensures space after comma in list (on)
e, p	**e** ensures space around =, and **p**, around + and – (on)
f	positions FORMAT statements just before the **END** (off)
j, t	**j** ensures space around ** and //, and **t**, around * and / (off)
k, o, q	**k** ensures space around **.AND.**, **.OR.**, **.EQV.**, **.NEQV.** (off); **o** ensures space around all logical operators (off); **q** is a combination of **k** and **o** which treats **IF** statements differently (on); only one of the three may be on
m	modifies spacing rules to permit short, otherwise two-line, statements to fit on one line (on)
n	ensures space around nonsubscript parentheses (off)
r	generates a block of comments listing externals (off)
s	applies **j**, **p**, **t** spacing rules to inside of parentheses (off)
x	shorthand specification of a popular style: format relabeling initialized at 900, other statement relabeling initialized at 100, both with an increment of 10, and insertion of comments summarizing externals (off)
y	beautifies only optimized blocks and echoes remainder of program (on)

These and other features can be selected and modified by the use of directives within the source files, applying to portions or to the entireties of the files. The **SWITCH** directive is the only way to change **FORMAT** and other statement label initial values and increments, indentations for different classes of statements, placement of statement labels, placement of comments, and the continuation line character. For example, the directives

```
CFPP$ SWITCH,FORMAT=900:10,RENUMB=100:10,LABELS=5:R,CONCHR=+
CFPP$ SWITCH,INDDO=3,INDIF=3,INDCN=3,LSTCOL=31
```

renumber **FORMAT** and other statement labels to starting values of 900 and 100, respectively, with increments of 10; right-adjust statement labels into column 5; specify "+" to be the continuation line character; specify that **DO** blocks and **IF** blocks be indented three columns for each nesting level; specify that continuation lines be indented three columns; and specify that column 31 be the last column available for indentation, i.e., that no statement begin beyond column 31. Except for the continuation line character, these illustrative values are the TIDY defaults.

Detailed instructions for the use of the TIDY function of **fpp** can be found in the Cray publication, *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0, pages 33 ff and 293 ff.

As stated previously, it is not quite possible to disable all but the TIDY functionality of **fpp**. One characteristic of **fpp** which cannot be disabled and which is highly visible to the user desiring ANSI Standard Fortran 77 output is the suffixion of certain Cray Fortran **INTRINSIC** subprogram names with an @ symbol. That this characteristic cannot be disabled is not addressed in the level 5.0 Cray documentation. This characteristic causes no difficulty at all, and, in fact, is a desired result, if the resulting code is then to be compiled by **cft77**, with or without intervening **fmp** processing. However, if the output is intended to be Fortran 77 source code with no greater degree of nonstandardness than the input, then the presence of the @ symbols is a corruption of the code.

There are several corrective strategies. One of the simplest is based on the realization that the @ symbols will almost certainly be sparse. In this case, it is practical to execute **grep –n @** on even large source files to identify and examine those lines containing @ symbols. With or without the **grep**, appropriate editors can be used to find and remove the offending @ symbols.

For those cases where the source code is so lengthy or so riddled with @ symbols that manual correction is undesireable, the **fpp** output can be piped through the filter of Figure 15.1 to remove exactly those @ symbols inserted by **fpp**.

15.3 Multitasking

Vector processing, discussed previously, is a form of parallel processing wherein sets of operands, rather than individual ones, are processed more or less at the same time. There are no overhead costs due to interprocess communication because only one CPU is used. Macrotasking, microtasking, and autotasking are techniques whereby multiple CPUs are used to expedite programs, vectorized or not, which otherwise take "too long" to complete. All three techniques are referred to as multitasking, and may be combined as desired within a program, given that the technique(s) chosen are usable with the programming language. All three techniques are usable with Fortran, but only microtasking and autotasking with C. There is no saving in terms of CPU time; indeed, a penalty is paid in terms of increased overhead. However, the wall clock time sometimes can be reduced considerably.

15.3.1 Macrotasking

Macrotasking was the first of the three techniques to be developed. It demands more of the programmer than the others, and often generates the most overhead. It applies multiple processors to a FORTAN job at the subroutine level. Macrotasking was designed for long running jobs, often in a dedicated environment. It is less useful than microtasking for small jobs because it has higher overhead. Macrotasking involves substantial changes to the program.

The process of macrotasking a program consists of program analysis and the insertion of certain subroutine calls to initiate tasks, wait for completion of tasks, synchronize tasks, handle locks used to control execution in critical regions of code, and modify tuning parameters within the library scheduler. A task is a Fortran entry point, typically a subprogram, and is normally completed when a **RETURN** or **STOP** is

executed. If an error occurs or a task aborts, then all tasks are stopped as quickly as possible.

Among the subprograms used are **BARrierASsiGN**, **BARRELease**, and **BARSYNChronize**; **EVentASGN**, **EVCLEAR**, **EVPOST**, **EVREL**, **EVTEST**, and **EVWAIT**; **LOCKASGN**, **LOCKOFF**, **LOCKON**, **LOCKREL**, and **LOCKTEST**; and **TaSKLIST**, **TSKSTART**, **TSKVALUE**, and **TSKWAIT**. Detailed information may be found in the on-line manual pages and in the Cray publication, *Volume 1: UNICOS Fortran Library Reference Manual*, SR–2079 6.0.

*****

```
gawk '
BEGIN { split("MOD@     ABS@     IABS@    INT@     REAL@"     \
              "MAX@     MAX0@    MIN@     MIN0@    AND@"      \
              "IAND@    OR@      IOR@     ISHFT@   SHIFTL@"   \
              "SHIFTR@  CVMGM@   CVMGN@   CVMGP@   CVMGT@"    \
              "CVMGZ@"                                       \
              "mod@     abs@     iabs@    int@     real@"     \
              "max@     max0@    min@     min0@    and@"      \
              "iand@    or@      ior@     ishft@   shiftl@"   \
              "shiftr@  cvmgm@   cvmgn@   cvmgp@   cvmgt@"    \
              "cvmgz@",                                      mod)
      n = split("MOD     ABS      IABS     INT      REAL"     \
              "MAX      MAX0     MIN      MIN0     AND"      \
              "IAND     OR       IOR      ISHFT    SHIFTL"   \
              "SHIFTR   CVMGM    CVMGN    CVMGP    CVMGT"    \
              "CVMGZ"                                        \
              "mod      abs      iabs     int      real"     \
              "max      max0     min      min0     and"      \
              "iand     or       ior      ishft    shiftl"   \
              "shiftr   cvmgm    cvmgn    cvmgp    cvmgt"    \
              "cvmgz",                                       orig)
    }

    { for (i = 1; i <= n; i++)
          gsub(mod[i],orig[i])
          print
    } '
```

Figure 15.1. Filter for Removing Suffixed @ Symbols in **fpp** Output

*****

15.3.2 Microtasking

Microtasking was developed after macrotasking to apply multiple processors to a program at the loop level. It demands much less of the programmer than does macrotasking and adds relatively little synchronization overhead.

The process of microtasking a program consists of program analysis and the insertion of certain directives which cause the compiling system to alter source code and invoke multitasking library routines as necessary. Because the directives appear to other compilers to be comments, microtasking does not reduce the degree to which a program conforms with the standard.

premult, not a part of the **cf77** compiling system, is a preprocessor whose task is to convert microtasking directives into invocations of the appropriate library routines. It will no longer be available when version

6.0 of the compiling system is released. Its functionality will be, and is currently, available from **fmp**, the Fortran multitasking translator. **fmp** is used also with autotasked code.

Autotasking is essentially enhanced microtasking performed automatically by the compiling system, with the opportunity for application of the programmer's insights. These insights are applied by inserting microtasking/autotasking directives into the source code. Microtasking and its directives are discussed further in the autotasking section.

Microtasked programs are compatible with autotasking. Detailed information may be found in the *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0.

15.3.3 Autotasking – fmp

Autotasking is the process, performed by the compiling system, of converting "normal," single processor Fortran source code, vectorized or not, into a form which will dynamically invoke multiple processors, based on the extent to which the program can make efficient use of them, and on the extent to which they are available.

The **fpp** preprocessor, discussed in the "Vectorization" section and invoked by **cf77 –Zv** or **cf77 –Zp**, not only analyzes source code and inserts directives which ultimately cause the **cft77** compiler to produce vectorized object code (and which influence its operation in other ways), but also analyzes source code and inserts directives which ultimately cause autotasked object code to be produced. (**fpp** always prefers vectorizing to autotasking.) With the **–Zv** option, the autotasking/microtasking directives are entirely ignored, the **cft77** compiler being able to interpret only the **CDIR$/CDIR@** directives.

When the compiling system is invoked with the **–Zp** option, the **fmp** preprocessor is invoked after **fpp** and before **cft77**. It is **fmp** which interprets all the autotasking/microtasking directives. The relationships among the preprocessors and directive types are as follows. All directives begin with a flag in column 1, which identifies them as directives or, when appropriate, as Fortran comments. Each flag has either a trailing $ or @ and is followed by at least one blank. $ indicates programmer insertion, while @ indicates insertion by **fpp** or **fmp**.

CFPP$ Directives inserted by the programmer to influence the operation of **fpp**.

CMIC$/@ Directives inserted by the programmer/**fpp** and interpreted by **fmp** to produce autotasked/microtasked object code. Some **CMIC$** directives inhibit **fpp**'s vectorization of certain portions of source code.

CDIR$/@ Directives inserted by the programmer/**fpp** and interpreted by **cft77** to produce vectorized code, or to have other effects. Certain **CDIR$** directives influence **fpp**'s operations.

15.3.4 User Interaction with Autotasking

User interaction with autotasking follows a sequence similar to that for vectorization, but has the potential to be considerably more complex. The preprocessors and compiler can produce output listings and intermediate source files (preprocessors only) which the programmer may examine and alter. Not only is there a complete suite of directives, but also **fmp** has a suite of options with which it may be invoked, either directly or by use of a **–Wu"..."** construct. Before embarking upon the effort of detailed interaction with the autotasking process, the programmer should consider whether the potential return is worth the cost. Detailed information, together with specific programming examples, may be found in the *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0.

15.3.5 Some Useful Utilities

The multitasking package maintains a history trace buffer containing the recent history of a multitasked program. This buffer is used by the following utilities.

mtdump Examines an unformatted dump of the multitasking history buffer. Generates reports according to its options.

multimeter Displays graphically the start/stop synchronization events in a multitasked program. Requires the X Window System.

stategraph Displays graphically the state transitions among tasks and processes in a multitasked program. Requires the X Window System.

timeline Displays graphically along a time line the connections and events among tasks and processes in a multitasked program. Requires the X Window System.

Intentionally Left Blank

16. Applications Software

16.1 The IMSL Library, Edition 10.0

The IMSL Library, Edition 10.0, has been installed on both Crays and is located in **/usr/local/lib/libimsl.a**. (See the chapter, "**cf77** Compiling System," for information about setting up library linkages.) The IMSL Library is organized into three main areas: general applied mathematics, statistics, and special functions. Most of the subprograms are available in both single and double precision versions. Contact **crayca@arl.army.mil** or (410) 278-6819/DSN 298-6819 for documentation.

The MATH/LIBRARY consists of hundreds of subprograms whose documentation is grouped into the following 10 chapters:

1. Linear Systems
2. Eigensystem Analysis
3. Interpolation and Approximation
4. Integration and Differentiation
5. Differential Equations
6. Transforms
7. Nonlinear Equations
8. Optimization
9. Basic Matrix/Vector Operations
10. Utilities

The STAT/LIBRARY consists of hundreds of subprograms whose documentation is grouped into the following 20 chapters:

1. Basic Statistics
2. Regression
3. Correlation
4. Analysis of Variance
5. Categorical and Discrete Data Analysis
6. Nonparametric Statistics
7. Tests of Goodness of Fit and Randomness
8. Time Series Analysis and Forecasting
9. Covariance Structures and Factor Analysis
10. Discriminant Analysis
11. Cluster Analysis
12. Sampling
13. Survival Analysis, Life Testing, and Reliability
14. Multidimensional Scaling
15. Density and Hazard Estimation
16. Line Printer Graphics
17. Probability Distribution Functions and Inverses
18. Random Number Generation
19. Utilities
20. Mathematical Support

The SFUN/LIBRARY consists of almost 200 subprograms whose documentation is grouped into the following 12 chapters:

1. Elementary Functions
2. Trigonometric and Hyperbolic Functions
3. Exponential Integrals and Related Functions
4. Gamma Function and Related Functions
5. Error Function and Related Functions
6. Bessel Functions
7. Kelvin Functions
8. Bessel Functions of Fractional Order
9. Elliptic Integrals
10. Weierstrass Elliptic and Related Functions
11. Probability Distribution Functions and Inverses
12. Miscellaneous Functions

16.2 The IMSL Library, Edition 9.2

The previous edition of the IMSL library, Edition 9.2, is no longer available on either Cray. There are substantial changes in subroutine names and arguments between editions 9.2 and 10.0. The file **/usr/pub/imsl.10** contains a list of edition 9.2 names and directs the user to the edition 10.0 subroutines by which they are replaced. It also contains information on the compatibility of the edition 9.2 and edition 10.0 subroutines.

16.3 LINDO – Linear, INteractive, Discrete Optimizer

Linear programming is a mathematical procedure for determining optimal allocation of scarce resources. LINDO is an interactive program used to formulate and solve linear programming problems. The Cray X–MP command line is **/usr/local/bin/lindo**.

16.4 PVI Graphics

Precision Visuals, Inc. (PVI) of Boulder, CO, provides a number of Fortran callable 2-D and 3-D graphics utilities including:

- DI–3000
- Grafmaker
- Contouring
- DI–Textpro
- GK–2000

Each of these graphics utilities interfaces with a common device driver library consisting of 50 different device drivers which support graphics terminals, pen plotters, laser printers, and color recording devices. The PVI graphics software has been installed on both the Cray–2 and the Cray X–MP. A complete list of the supported devices can be found in **/usr/pvi/bin/DRIVER–LIST**.

PVI also provides two interactive graphics utilities:

- PicSure/Plus
- Metafile/CGM Translator

These two packages interface with the same device driver library as the Fortran callable packages. The PicSure library is an interactive package invoked from the shell. The metafile library is available both to Fortran programs and by use of the interactive Metafile Translator. A metafile is a sequential file which contains a series of graphics images.

Documentation for the PVI products includes:

- DI–3000 User's Guide and Quick Reference Guide
- Grafmaker/Grafeasy User's Guide and Quick Reference Guide
- Contouring System User's Guide
- GK–2000 User's Guide and Quick Reference Guide
- Metafile Translator User's Guide
- PicSure User's Guide and Quick Reference Guide
- PicSure Chart Book

16.4.1 Fortran Callable Subroutines

16.4.1.1 DI–3000

This is a very flexible, low level graphics library which supports full device color, 2-D and 3-D primitives, shaded pattern areas, full graphics input, real time image manipulation, and object modeling. This library can be used to develop customized graphics applications in which the programmer needs precise control over the characteristics of the final graphical output. DI–3000 consists of over 200 user-callable subroutines. An executable created with the DI–3000 library is output device independent; that is, changing an environment variable is sufficient to cause the output to be sent to a different device and to utilize fully that device's capabilities. See the following section, "PVI Utilization," for more information.

16.4.1.2 Grafmaker and Grafeasy

This library contains high level Fortran callable graphics subroutines which, in turn, invoke DI–3000 subroutines. It can be used for building routine applications such as might produce line graphs and bar and pie charts.

Although not as flexible as DI–3000, this library permits programs producing routine data presentation output to be developed quickly. If more precise control over certain chart attributes is needed, the lower level DI–3000 subroutines can be invoked explicitly along with the Grafmaker subroutines.

Grafeasy is a related product. It creates rudimentary data presentation charts and provides minimal programming options. This product can be very useful for black and white terminal display images.

16.4.1.3 DI–Textpro

This option to DI–3000 invokes a library of high precision, presentation quality fonts. Some of the fonts available follow, in outline form. Variations, including italics, are available.

Swiss
Swiss Light
Swiss Bold
Humanist Bold
Aldine
Stencil

Aldine Classic Italic
Aldine Classic Extra Italic
Century Bold
OCR
Geometric

16.4.1.4 Contouring

Contouring library subroutines may be invoked from within DI–3000 based programs. This Fortran library is used to create mesh or contour map surfaces from gridded or random data. The contouring system provides monochrome and color contours, color-filled maps, 3-D contour lines, 3-D mesh surfaces, hidden line removal, 3-D axes, and marker/annotation control. Because the contouring library invokes DI–3000 subroutines, the programmer has complete control over chart features and device independent graphic output.

16.4.1.5 GK–2000

This library, consisting of over 200 user-callable Fortran subroutines, is completely separate from DI–3000. The GK–2000 package is certified for compliance with the Graphical Kernel System (GKS) level 2b specifications. GKS is a world-wide, machine and device independent graphics standard for 2-D graphics. GKS compliance means that the set of subroutine invocations used to produce graphics output is standardized. The standard specifies subroutine names, arguments, and actions. Therefore, this software is particularly valuable for writing portable code.

16.4.2 PVI Utilization

16.4.2.1 Environment Variables

A number of environment variables must be set before using the PVI software:

For **/bin/sh** users, in the **.profile** file, or elsewhere:

```
PVL_ROOT=/usr/pvi
PVL_LINK=dynamic
PVL_DEV_1=xxx
[PVL_DEV_n=xxx]
[...]
PVL_LDFLAGS=filename
PATH=$PATH:$PVL_ROOT/bin
export PVL_ROOT PVL_LINK PVL_DEV_1 [...] PATH
```

For **/bin/csh** and **/bin/tcsh** users, in the **.login** file, or elsewhere:

setenv PVI_ROOT /usr/pvi
setenv PVI_LINK dynamic
setenv PVI_DEV_1 *xxx*
[**setenv PVI_DEV_***n xxx*]
[...]
setenv PVI_LDFLAGS *filename*
set path=($path $PVI_ROOT/bin)

PVI_DEV_*n* specifies a location where temporary loader information can be written. A conventional practice is to use **/tmp/***username* for *filename*. *xxx* is a 3-character code for the device driver to be used as the output device. PVI programs may use up to eight different output devices concurrently. The devices are assigned by environment variables **PVI_DEV_***n*, *n* from 1 through 8, inclusive. **PVI_DEV_1** is required and is the primary output device; the others are optional.

16.4.2.2 Compiling and Loading PVI Programs

The following commands will compile and load PVI programs. Both scripts invoke the **cft77** compiler, followed by the **segldr**. To create an executable using DI–3000, Grafmaker, DI–Textpro, and contouring, use:

di3load [–SS] [–HP] [–HLR] [–MF] [–GM] [–TP] [–CN] [*segldr_options*] *filenames*.**f**

where the options specify

–SS	segment storage library
–HP	high precision library
–HLR	hidden line removal
–MF	metafile library
–GM	Grafmaker/Grafeasy library
–TP	DI–Textpro library
–CN	contouring library

To create an executable using GK–2000, use:

gk2load [–MF] [–SS] [*segldr_options*] *filenames*.**f**

where the options specify

–MF	metafile library
–SS	segment storage library

16.4.3 Interactive Graphics Applications

16.4.3.1 PicSure/Plus

PicSure is an interactive computer graphics software system for generating 2-D charts and graphs with simple sequences of English-like commands. Little knowledge of programming is required. This package can be used to present data in many different forms. The basic PicSure chart types are line graphs and bar, pie, and text charts. Variations of the basic chart types, such as stacked line graphs, horizontal bar charts, stacked bar charts, and scattergrams are available. Multiple smaller charts can be combined into a single composite chart.

Usually, PicSure is run as an interactive program, but it can accept commands from an external file, as if they were being entered in sequence from the keyboard. The user enters a sequence of commands to build a chart, which then is drawn on one or more (up to eight) graphics display devices, or is stored in a metafile. Any device supported in the common device library can be used with PicSure. See the following sections on Metafile/CGM for more details. The *PicSure User's Guide* provides an excellent introduction and tutorial on the use of PicSure and is strongly recommended for first time users.

16.4.3.2 Using PicSure

A number of environment variables must be set before using PicSure.

For **/bin/sh** users, in the **.profile** file, or elsewhere:

```
PVI_ROOT=/usr/pvi
PATH=$PATH:$PVI_ROOT/bin
export PVI_ROOT PATH
```

For **/bin/csh** and **/bin/tcsh** users, in the **.login** file, or elsewhere:

```
setenv PVI_ROOT /usr/pvi
set path=($path $PVI_ROOT/bin)
```

To invoke PicSure, execute

picsure [*drv1* [... *drv8*]]

where each of the *drvn* is a 3-character code for the device driver to be used as an output device, as discussed previously in this chapter. If no arguments are specified, a prompt will be issued for at least one.

16.4.3.3 Metafile/CGM Translator

The Metafile System provides a method for storing graphical information from an application as an external file which is both device and machine independent. The external file, called a metafile, is a sequential file of images created by graphics software. DI–3000, GK–2000, and PicSure can produce metafiles. The images in a metafile are created for a virtual graphics device and become associated with a physical device only at the time of actual rendering of the graphics output into an image.

A Computer Generated Metafile (CGM) is a different kind of metafile. It conforms to the first international standard for metafile images and is not only machine and device independent, but also is independent of the application which produced the image.

16.4.3.4 Using Metafile/CGM Translator

To use the metafile system, the user must first establish the appropriate environment variables, as discussed previously for other PVI software. There are two different procedures available for viewing metafiles. The first,

mftran [*options*] [*drv1* ...]

is used to view PVI format metafiles, to produce additional PVI format metafiles, or to translate such files to the CGM format. The other,

cgmint [*drv1*...]

is used to convert CGM metafiles to a particular graphics output device format or to the PVI metafile format. *options* have to do with conformance to the CGM standard and *drv1* ... is a list of up to eight device driver codes, as discussed previously.

16.5 CA–DISSPLA 11.0 Graphics Library

DISSPLA (Display Integrated Software System and Plotting LAnguage) is a high level Fortran graphics subroutine library. The library on the Cray X–MP includes over 400 Fortran subroutines and functions which can produce general purpose 2-D and 3-D graphics, and three option packages which can produce, for example, maps with correct political boundaries, calendar axes, and applications conforming to the international standard Graphical Kernel System (GKS). CA–DISSPLA/CA–GKS supports over 300 different graphics devices. Basic instructions on the use of a particular device are provided in the directory **/usr/local/dissplа11/doc**.

To be compatible with CA–DISSPLA or CA–GKS, the application program must be written in a language compatible with Fortran. The application is then compiled and linked with the appropriate object libraries:

/usr/local/dissplа11/lib/libdcc.a
/usr/local/dissplа11/lib/libdis77.a
/usr/local/dissplа11/lib/libdisman77.a
/usr/local/dissplа11/lib/libgks.a
/usr/local/dissplа11/lib/libint.a

to produce an executable program. **cf77** and **segldr** can be invoked explicitly with one or more **–l** options, or their equivalents, or one of the compiler/linker scripts provided with the CA–DISSPLA package can be used. To use the CA–DISSPLA linker, enter **/usr/local/dissplа11/dis77link** *program*, where *program* is the base portion of the filename, *program*.**f**. To use the GKS linker, enter **/usr/local/dissplа11/gkslink** *program*. Each linker produces an executable named *program*.

The environment variable **SFDATA** must be set prior to executing the program. For **/bin/sh** users, in the **.profile** file, or elsewhere:

```
SFDATA=/usr/local/dissplа11/data/dsdf.dat
export SFDATA
```

For **/bin/csh** and **/bin/tcsh** users, in the **.login** file, or elsewhere:

```
setenv SFDATA /usr/local/dissplа11/data/dsdf.dat
```

16.6 CA–DISSPLA 10.0 Graphics Library

CA–DISSPLA 10.0 is the release level running on the Cray–2, probably until mid-1993, at which time level 11.0 is scheduled to become the released version for the Cray–2. Level 10.0 functionality is the same as that of level 11.0, discussed previously. Library names, locations, and linkages, however, are different, as follows:

/usr/local/lib/dissplа.10/dcclib.a
/usr/local/lib/dissplа.10/dislib.a
/usr/local/lib/dissplа.10/gkslib.a
/usr/local/lib/dissplа.10/intlib.a
/usr/local/lib/dissplа.10/pvilib.a

Special linker programs are not provided in CA–DISSPLA 10.0. The libraries must be linked with a user's object files using a special **segldr** command, **segld5**, which is designed to reconcile incompatibilities between the CA–DISSPLA 10.0 libraries and the default scientific libraries used by the UNICOS 6.0 **segldr**. A user must do a separate compilation, **cft77**, followed by a **segld5** invocation, instead of using the **cf77** compiling system which automatically calls the normal **segldr**. **segld5** has the same syntax as **segldr**; therefore, the CA–DISSPLA 10.0 libraries may be specified in the usual manner. It is not necessary to set an environmental variable corresponding to version 11.0's **SFDATA**.

16.7 MPGS

MPGS (MultiPurpose Graphics System) is a Cray Research, Inc. product which interactively performs distributed visualization postprocessing of data files resident on a Cray supercomputer. It is discussed in the chapter, "Scientific Visualization."

16.8 BRL–CAD

BRL–CAD (Ballistic Research Laboratory CAD) is a large, interactive, public domain, combinatorial solid geometry (CSG) based modeling system written entirely in-house at the then USABRL. It is discussed in the chapter, "Scientific Visualization."

16.9 BRLLIB

BRLLIB is a collection of Fortran utility subprograms formerly available on BRLESC I and II (pronounced "burlesque," and meaning Ballistic Research Laboratory Electronic Scientific Computer; computers built in-house during a time when BRL could and did build better machines than were available in the marketplace) and then on the BRL CYBER computers under the collective name, "BRLLIB." In large part, these subprograms have been in more or less continuous use by scientists and engineers of the former BRL for over 25 years. They are not provided as a library, but as a collection of separate source code files, **/usr/local/brllib/subprograms/*.f** on the Cray X–MP, which can be incorporated into a user's programs. The **man** page gives a description of the subprograms available. Each subprogram contains more complete internal documentation. In addition, more complete documentation for all of the subprograms is collected into the file **/usr/local/brllib/descriptions**.

16.10 MR – Stepwise Multiple Regression

The BRL Stepwise Multiple Regression Program, named MR, was written long ago at BRL for use on BRLESC I and II (pronounced "burlesque," and meaning Ballistic Research Laboratory Electronic Scientific Computer; computers built in-house during a time when BRL could and did build better machines than were available in the marketplace) in the FORAST language (an early language more or less similar to Fortran). The program was subjected to any number of rewrites/patches to cast it into various nonstandard versions of Fortran, to make it run on newer machines, and to introduce additional functionality. MR source code is available on the Cray X–MP as the collection of **.f** files in **/usr/local/brllib/mr**.

The MR version currently in use:

- supersedes all earlier versions.
- runs on many ARLSCF machines, including both Crays.
- is entirely in accord with the ANSI Fortran 77 standard, except that, for legibility, the code is entirely in lower case characters, except certain constants.
- is more modular and structured than its predecessors.
- takes advantage of the UNIX-based computing environment to make input and output much more convenient than its predecessors.

- does not have its predecessors' "rescale" capability, which capability is better implemented by the user as a part of the physical phenomenon description in the **form** subroutines.
- uses PVI GRAFMAKER software and the GRAFMAKER – UNIX interface to produce plots equivalent to those of its predecessors.

The MR program is extremely general in that it can be applied to any physical phenomenon; there is, however, a cost associated with this generality. The MR source code proper contains all the mathematics required for the regression analysis, but the modeling specific to the details of the physical phenomenon of interest is absent. The user must encode in ANSI Standard Fortran 77 subroutines (and, if desired, in other subprograms invoked by them) the details of the physical relationships thought to exist within the data being regressed.

MR source code is available on the Cray X–MP as the collection of **.f** files in **/usr/local/brllib/mr**. Of those **.f** files, the ones with base names **blockdata**, **dropg**, **form1**, **form2**, **form3**, **form4**, **form5**, **form6**, **form7**, **form8**, **form10**, **liofst**, and **prepar** are not a part of MR proper, but rather address certain exterior ballistics analyses, and are provided merely as examples. File **form9.f** is a skeleton displaying the minimum requirements of the **form** subroutine interface. An executable is not provided because certain environmental variables required by the GRAFMAKER – UNIX interface must be available at load time (the variables' values change from one user to another), and, of course, because different uses require the incorporation of different **form** subroutines. Hence, before his first execution of MR, each user must define his own set of environmental variables and create his own PVI output device configuration file (if the graphics output capabilities are to be used). Before the first MR analysis of a given phenomenon, a user must encode the **form** subroutines modeling that phenomenon, compile the collection of **.f** files proper to MR along with the ones specific to the phenomenon of interest, and load the resulting **.o** files and appropriate libraries to obtain an executable. The load sequence required for PVI software is intricate and has been encapsulated in the script **di3load**.

16.11 SIMSCRIPT II.5

SIMSCRIPT II.5, a language designed specifically for simulation, is available on the Cray–2.

The SIMSCRIPT II.5 compiler, **/usr/local/bin/simc**, translates a program written in the SIMSCRIPT II.5 programming language into C, invokes **cc** to produce one or more **.o** files, and, by default, invokes **segldr** to produce the executable, **a.out**. The SIMSCRIPT II.5 loader, **/usr/local/bin/simld**, invokes **segldr** to load the specified **.o** files and appropriate objects from libraries to produce the executable, **a.out**. By convention, files containing SIMSCRIPT II.5 source code are given names that end in **.sim**.

To compile a SIMSCRIPT II.5 program, enter:

simc [*options*] *files*.**sim**

where the *options* are:

–c Suppresses loading and produces a **.o** file with the same base name as the SIMSCRIPT source file.

–C Generates additional code to perform run time checking of every array element reference and every attribute reference; this is recommended until the program is fully debugged.

–f Produces FLOWTRACE profiling information during execution.

–g Generates tables associated with each routine to provide a more elaborate traceback of the program state in the event of a run time error or TRACE.

–l Writes a listing to standard output which includes the source statements and any diagnostic messages and, if **–x** or **–X** have been specified, the local and/or global cross references.

–o *executable* When an executable is produced, name it *executable* instead of **a.out**.

–S Produces file(s) containing the generated source code for the routine together with the SIMSCRIPT source code as comments.

–v Inhibits the listing of the preamble and the generation of any "scripted" routines used to perform such things as set management and entity creation and deletion.

–w Suppresses warning messages.

–x Writes to the listing the local cross references for each routine.

–X Writes to the listing the global cross references for each routine.

To produce an executable from existing SIMSCRIPT II.5 object (**.o**) files, enter **simld** [**–o** *name*] *files*.**o**, where **–o** *name* specifies a *name* other than **a.out** for the executable.

SIMSCRIPT II.5 executables are executed in the usual manner. Parameters specified on the command line are available to the program in the SIMSCRIPT II.5 global TEXT array, PARM.V. For example, given the command:

```
a.out –j 5 xyz.data
```

the PARM.V array is initialized as follows (SIMSCRIPT II.5 source code):

```
DIM.F(PARM.V(*))=3
PARM.V(1)="–j"
PARM.V(2)="5"
PARM.V(3)="xyz.data"
```

A SIMSCRIPT II.5 program uses unit 5 for standard input, unit 6 for standard output, and unit 98 for standard error. I/O involving these units may be redirected in the usual way.

16.12 LQGALPHA

LQGALPHA is a library of Fortran programs for the development and analysis of linear multivariable control system designs (both continuous and discrete time) based on the Linear–Quadratic–Gaussian (LQG) design and singular value analysis methodologies. LQGALPHA can be used to

- design linear feedback systems
- analyze linear systems using classical tools such as Bode plots, Nyquist plots, Inverse Nyquist plots, and Nichols charts
- analyze linear systems using modern singular value analysis
- simulate deterministic and stochastic linear systems

LQGALPHA is available on the Cray X–MP. Additional information and documentation may be obtained from the licensor,

ALPHATECH, Inc.
2 Burlington Executive Center
111 Middlesex Turnpike
Burlington, Massachusetts 01803
1-617-273-3388

16.13 PROLOG

Prolog (PROgrammation en LOGique) is a nonprocedural programming language based on the first order predicate calculus. J. W. Lloyd's *Foundations of Logic Programming* (Springer–Verlag 1985) provides a description of the mathematical underpinnings of Prolog and other logic programming systems. The

language was designed by Alain Colmerauer of the University of Marseilles and originally implemented there by Philippe Roussel in 1972. Prolog saw little use outside of European universities until 1980, when it was chosen as the implementation language for the Japanese Fifth Generation computer project. Since then, it has grown in popularity as a language for prototyping, problem specification, expert systems development, and research in artificial intelligence.

Cray Prolog, available only on the Cray X–MP, was designed and developed at Cray Research by Peter Klausler. It uses several novel techniques, including partial clause compilation and vectorized structure copying to provide efficient execution of logic programs on Cray Research supercomputers.

Prolog is described in the book *Programming in Prolog* by W. Clocksin and C. Mellish. Cray Prolog differs from Clocksin and Mellish at least in that, in Cray Prolog:

- It is not possible to find more than one outcome in a search. This is a fatal problem.
- Periods terminate clauses if and only if they are followed by white space characters, are not preceded by another period, and are not nested within braces, brackets, or parentheses, with the following exception.
- Integers at the end of a query are not parsed correctly in that ". " after an integer is parsed as a decimal point.
- Cray prolog thinks [] is not atomic; i.e., **atomic([]).** fails when it should succeed.
- Atoms defined to be prefix operators may not be used as functors.
- At end of file, **read=** will forever return the term **?–end**.
- The predefined predicate **clause** is defined as **clause** (*X*), where *X* unifies with a clause in the database.
- The predefined predicate **debugging** is defined as **debugging** (*X*), where *X* matches a list of active spy point specifications.
- **compile**, **once**, **dump**, and **system** are predefined predicates.
- **write**, **display**, and **listing** print unbound variables with their original names.
- The **NOLC** and **LC** predefined predicates are not supported.
- The operators "**not**" and "**,**" have slightly different signatures; "**not**" is "*fy*, **not** (*fx*)", and "**,**" is "*xfy*, **not** (*yfx*)".

16.14 SCIPORT

The SCIPORT library, available on the Cray X–MP as /usr/local/sciport, is a portable implementation of the Cray SCILIB Mathematical Library. It was developed by General Electric Corporate Research and Development to enhance convertibility between local computer systems, where applications are developed, and the Cray environment, which is suited to production use. The SCIPORT library is a collection of portable utility subroutines encoded in Fortran. SCIPORT reproduces the functions of corresponding routines in Cray Research's SCILIB. Thus, the user developing applications locally can use SCILIB calls during program development with the assurance that results will be consistent (within floating point precision) between the local computer and the Cray.

In addition, these portable, compact, and efficient SCIPORT routines provide a convenient source for commonly used mathematical software, such as the BLAS (Basic Linear Algebra Subroutines), which occur frequently in higher level libraries like LINPACK, EISPACK, and others.

16.15 ABAQUS

ABAQUS, available on the Cray X–MP, is a general-purpose finite element analysis program for use in the numerical modeling of structural response. Stress problems can be divided into two types, static and dynamic response, depending upon whether inertial effects are significant. ABAQUS permits the same analysis to be used for both the static and dynamic phases. Procedures are available within ABAQUS for the following:

- static stress analysis
- dynamic analysis
- heat transfer
- element removal/replacement
- coupled pore fluid diffusion and stress
- eigenvalue buckling prediction
- coupled heat transfer/stress analysis
- natural frequency extraction
- J–integral evaluation
- geostatic stress state
- dynamic analysis of linear systems by modal methods
- substructuring/superelements
- loading specification
- rezoning
- acoustic and coupled acoustic-structural analysis

The scripts

/usr/local/abaqus/bin/abaqus
/usr/local/abaqus/bin/abaplot
/usr/local/abaqus/bin/abapost

constitute the user interface with ABAQUS. Documentation can be obtained from

Hibbitt, Karlsson & Sorensen, Inc.
1080 Main Street
Pawtucket, RI 02860
1-401-727-4200

16.16 MSC/NASTRAN

This software is not accessible to users pending availability of funds to meet licensing costs. User organizations willing to provide partial funding should contact the Chief, High Performance Computing & Communications Branch, listed in Appendix B.

MSC/NASTRAN is a large-scale, general-purpose digital computer program which can be used to solve a wide variety of engineering problems by the finite element method. It has been developed and is maintained by the MacNeal–Schwendler Corporation (MSC) from the original NASTRAN general-purpose structural analysis program of the National Aeronautics and Space Administration (NASA). Executables and example data and result sets are in directory **/usr/src/src3d/Nastran** on the Cray X–MP.

MSC/NASTRAN is invoked by

/usr/local/bin/nastran [**jid=**]*input-file-base-name* [*options*]

Options are discussed in the **man** pages and in MSC documentation.

The ARLSCF has only reference copies of the MSC documentation. Organizational/personal copies of the documentation may be purchased directly from MSC. Call the Product Service Center at 1-800-336-4858 for current Government pricing. Purchase orders should be sent to:

MacNeal–Schwendler Corporation
Product Service Center
815 Colorado Blvd.
Los Angeles, CA 90041-1777

16.17 MSC/DYNA

This software is not accessible to users pending availability of funds to meet licensing costs. User organizations willing to provide partial funding should contact the Chief, High Performance Computing & Communications Branch, listed in Appendix B.

MSC/DYNA is a 3-D finite element code for analyzing the dynamic, nonlinear behavior of solid components and structures. It uses explicit time integration and incorporates features to simulate a wide range of material and geometric nonlinearity. It is particularly suitable for analyzing transient events that incorporate a high degree of nonlinearity. Executables and example data and result sets are in directory **/usr/src/src3d/Dyna3** on the Cray X–MP.

MSC/DYNA is invoked by

/usr/local/bin/dyna [**jid=**]*input-file-base-name* [*options*]

Options are discussed in the **man** pages and in MSC documentation.

The ARLSCF has only reference copies of the MSC documentation. Organizational/personal copies of the documentation may be purchased directly from MSC. Call the Product Service Center at 1-800-336-4858 for current Government pricing. Purchase orders should be sent to:

MacNeal–Schwendler Corporation
Product Service Center
815 Colorado Blvd.
Los Angeles, CA 90041-1777

16.18 MSC/PISCES

This software is not accessible to users pending availability of funds to meet licensing costs. User organizations willing to provide partial funding should contact the Chief, High Performance Computing & Communications Branch, listed in Appendix B.

MSC/PISCES 2–DELK is a 2-D explicit finite difference computer program used to study the response and interaction of fluids and solids to static or dynamic loads.

Shell scripts, executables, and example data and result sets are in directory **/usr/src/src3d/Pisces.v30** on the Cray X–MP.

The ARLSCF has only reference copies of the MSC documentation. Organizational/personal copies of the documentation may be purchased directly from MSC. Call the Product Service Center at 1-800-336-4858 for current Government pricing. Purchase orders should be sent to:

MacNeal–Schwendler Corporation
Product Service Center
815 Colorado Blvd.
Los Angeles, CA 90041-1777

16.19 MSC/DYTRAN

This software is not accessible to users pending availability of funds to meet licensing costs. User organizations willing to provide partial funding should contact the Chief, High Performance Computing & Communications Branch, listed in Appendix B.

MSC/DYTRAN version 2 is a 3-D finite element code particularly suitable for analyzing transient events involving large deformations, a high degree of nonlinearity, and interactions between fluids and structures. Lagrangian and Eulerian processors are available.

Shell scripts, executables, and example data and result sets are in directory **/usr/src/src3d/Dytran** on the Cray X–MP.

The ARLSCF has only reference copies of the MSC documentation. Organizational/personal copies of the documentation may be purchased directly from MSC. Call the Product Service Center at 1-800-336-4858 for current Government pricing. Purchase orders should be sent to:

MacNeal–Schwendler Corporation
Product Service Center
815 Colorado Blvd.
Los Angeles, CA 90041-1777

17. Scientific Visualization

17.1 Introduction

Scientific visualization is the rendering of scientific data as images. Its end product is often a video animation, but it encompasses still image production and interactive data exploration. The data are often the result of computations performed within a grid system, but data from other types of computation and experimental data also can benefit from visualization techniques.

Scientific visualization has two major purposes: to assist scientists and engineers in their work, and to explain and promote the science or engineering discipline to others. The requirements for the graphics used in these two situations often differ. In the former, the graphics must facilitate detailed examination of the data, while in the latter, the graphics must be attractive and easily understood by the uninitiate.

Scientific visualization provides a capability to present traditional data in nontraditional and innovative ways. For example, computer solutions of difficult or intricate problems often produce immense quantities of numerical data, quantities so large as to defy attempts at synthesis to obtain understanding of the phenomena. For years, such large data sets routinely have been rendered as X–Y plots or contour plots on monochrome terminals and pen plotters. Today, software and hardware exist to present the same information as 3-D, light source shaded, colored surfaces, conveying more information in a much more readily apprehended manner. In addition, certain variables (e.g., time, speed, fraction of a particular constituent in an alloy or chemical mixture) may be parameterized to produce sets of individual solutions. The individual solutions can be visualized and then displayed sequentially to achieve animation, where the "action" highlights dependence upon some variable of particular interest. Visualizations, static or dynamic, can be viewed from different directions and zoomed to provide an overall appreciation of the data set in its entirety or very close examination of its smallest details.

These capabilities for overall synthesis and almost simultaneous differentiation of detail are provided by no technique other than visualization, and they can improve dramatically an investigator's insight into a problem.

17.2 Resources

Several organizations have combined their resources, facilities, and equipment to create the unclassified visualization laboratory located in Building 390, Room 133. This facility, although of limited capacity, is available to interested parties for visualization and animation production work. A Silicon Graphics (SGI) 440VGX with approximately 10 GB of disk storage is the central computing server, and there is an SGI 420VGX with approximately 20 GB of high-speed disk storage for special applications. In addition, facility personnel are available to assist organizations with large requirements in developing their own visualization capability.

A similar cooperative effort has resulted in the classified animation facility in the building 309 computer site. An SGI 320VGX system and miscellaneous video equipment have been procured and soon will be available for general use.

The Scientific Visualization Team periodically sponsors seminars and open houses to announce and demonstrate new resources, to educate users, and to demonstrate visualization techniques. For additional information and assistance, contact the team via electronic mail at **vis@arl.army.mil**.

17.3 Hardware

Currently, the hardware inventory for the two visualization facilities is:

- SGI 4D/440VGX (64 MB memory; 8, 1.2 GB SCSI drives; multibuffer)
- SGI 4D/420VGX (64 MB memory; 8, 3 GB IPI drives; multibuffer)
- SGI 4D/320VGX (64 MB memory; 1, 1.1 GB IPI drive)
- Abekas A60 networked digital video disk
- Abekas "Solo" A34 Video Mixer and Production System
- Chromatek 9120 scan converter
- Faroudja RGB to NTSC encoder
- Tektronix TSG–170A synchronization generator
- Tektronix 1870R measurement set
- Panasonic TQ–3031F video disk player/recorder
- Panasonic TQ–3032F video disk player
- Sony BVU–870 U–matic (3/4 inch) SP VCR
- Sony VO–9600 U–matic (3/4 inch) SP VCR
- Lyon–Lamb MiniVas VTR controller
- Spatial Systems Spaceball

17.4 Software

Because of its complexity, the visualization software cannot be described in detail in this document. More complete descriptions and usage information than are presented herein can be obtained from the Scientific Visualization Team (email: **vis@arl.army.mil**). In general, the software packages are not run on the ARLSCF Crays (MPGS, whose description follows, is an exception); instead, they are used to postprocess large amounts of data often associated with applications running on the Crays. The software inventory is expanding as the Scientific Visualization Team investigates new software packages which have the potential to meet the needs of the scientific computing community. The scientific visualization packages of primary interest are:

- MPGS (MultiPurpose Graphics System)
- BRL–CAD
- Advanced Visualizer
- Data Visualizer
- PV–Wave
- BRL–ShAYD

17.4.1 MPGS – MultiPurpose Graphics System

MPGS is a Cray Research, Inc. product which interactively performs distributed visualization postprocessing of data files resident on a Cray supercomputer. It is distributed between a Cray running UNICOS and

UNIX-based workstations, either Silicon Graphics workstations in the IRIS–4D series, including Indigo, or IBM RS6000 workstations with graphics upgrades. Communication between the workstation and the supercomputer is effected via the TCP/IP network protocol. Two versions of MPGS are available on each ARLSCF Cray, **/usr/local/bin/mpgs3.5** and **/usr/local/bin/mpgs4.1**, to accommodate the operating system level of the workstation involved. There is no charge for installing the workstation portion of MPGS.

The Cray system handles memory-intensive and CPU-intensive tasks, and the workstation, local graphics manipulations such as rotation, zoom, hidden line removal, and key-frame animation. This workload distribution and the continued residence of the dataset on the Cray ensure the efficient use of both computer systems and minimize network data transfers.

MPGS is useful for the visualization of data in almost any scientific or engineering area, e.g., structural analysis, computational fluid dynamics, electromagnetics, and thermodynamics. Its key features are:

- interactive, menu driven
- has a flexible parts structure
- supports dynamic transformations
- supports line drawings and hidden line drawings
- supports false color and shading
- can display all nodes in a computational mesh
- produces contours (isolines) of equal scalar magnitude
- processes scalar, vector, and discrete particle data
- supports arbitrary 2-D clipping planes

MPGS input files must be in the MPGS file format, or in the Movie–BYU format. Converters are available for several popular data file formats such as Dyna3D, Nastran, Patran, and Plot3D. Generally, these interactive data converters prompt the user for information about the input file and create the appropriate MPGS output file.

17.4.2 BRL–CAD

BRL–CAD is a large, combinatorial solid geometry (CSG) based modeling system written entirely in-house at the then USABRL. It is in the public domain and is installed on the two ARLSCF Crays and various other ARLSCF machines. Its main components are a solid model editor (MGED), a ray tracing library for model interrogation (librt.a), a generic frame buffer library with full network display capability (libfb.a), and a large collection of software tools for frame buffer and image manipulation and analysis. These utilities, augmented by in-house format converters, color mappers, etc., have been a significant component of visualization work to date.

Documentation for BRL–CAD is available from the ARLSCF staff, in the **man** pages, and in the USABRL document entitled *The Ballistic Research Laboratory CAD Package.*

17.4.3 Wavefront: Advanced Visualizer

This software is licensed for individual Silicon Graphics workstations and is available on several SGIs in the visualization facility. It is used routinely for geometric modeling applications such as sabot and penetrator design, and it provides tools required to generate 3-D geometry and features such as surface materials and textures. Data converters are available to read both IDEAS and Movie–BYU formatted geometry files into the Advanced Visualizer.

The Advanced Visualizer consists of four modules:

Model This module builds or imports objects. It provides the tools required to generate 3-D geometry and to apply materials and textures to the generated surfaces.

Preview The animation module. It is used to produce complicated motion from multiple objects, cameras, and lights.

Medit This module is used to create and edit materials, textures, colors, atmospheres, lights, and reflections. This information is passed to Model which creates a library of surfaces with these characteristics.

Image This module outputs a finished image to a file or output device. Multiple individual files each contain a single time step of an animation; when viewed sequentially, they provide the animation of the described object.

17.4.4 Wavefront: Data Visualizer

Wavefront's Data Visualizer is a 3-D volumetric visualization package. It is useful for animating recorded or numerically simulated data. It consists of numerous tools which permit the visualization of scalar and vector fields and any associated meshes. The graphics and user interfaces are separate from the data server, so remote data (e.g., resident on the Cray X–MP) can be viewed on a local SGI workstation.

Third-party data converters are available to convert files into a format compatible with the Data Visualizer. General purpose formats which can be so converted include Plot3D, HDF, AVS, and Movie–BYU. HULL, AVCO, USA–PG3 and Dyna3D converters have been written locally.

17.4.5 Precision Visuals: PV–Wave

PV–Wave is workstation-based software which permits interactive exploration and visualization of large datasets. This software is particularly useful for interactive data reduction and filtering, 2-D plotting (e.g., time series graphs, polar plots, and log/linear charts), 3-D data display (e.g., contouring, hidden line mesh surfaces, and light source shading), and image analysis and manipulation. PV–Wave is designed specifically for X–windows output and can be run from **image.arl.army.mil** (VAX 6320), with graphics output being sent to any X–window device.

This software can be licensed to run on SGI, Sun, DEC, HP, and IBM workstations, and is available in a command language interface version and in a window-based point-and-click interface version.

17.4.6 BRL–ShAYD

BRL–ShAYD (Shaded polygons At Your Desk) is a locally developed package for software rendering of data in the Plot3D format, with the output displayed either on an X–window device or a BRL–CAD frame buffer. ShAYD consists of four parts:

- A grid and solution server which reads data files from the platform where the computation was performed.
- A viewing pipeline which renders the grids and solutions as 3-D Gourand shaded polygons.
- A simple command driven user interface which controls the viewing pipeline and data server.
- A MOTIF-based graphical user interface (GUI) which provides a point-and-click X–window interface to the command language.

BRL–ShAYD is particularly useful for viewing very large datasets which may be too large for some of the workstation-based packages. ShAYD permits examination of these datasets by taking "slices" of the data

and displaying only those slices rather than trying to render the entire dataset.

The polygon rendering performed by ShAYD is accomplished in software rather than by relying on specific workstation hardware. Therefore, users with access to relatively inexpensive X–window-based terminals and workstations which do not provide graphics capabilities in hardware can use this software package.

17.4.7 New Software

Several new packages whose utility and reliability have not yet been fully defined are available.

SciAn, a public domain scientific visualization and animation program for graphics workstations, is very promising. It is being developed at Florida State University and the Supercomputer Research Institute, partially funded by a Department of Energy grant. A beta test version is available on Silicon Graphics workstations. Currently, SciAn reads only HDF (Hierarchical Data Format) files, but plans for the future include a distributed file reader with a Plot3D translator.

A software package called n–Title is available to generate animated text on Silicon Graphics workstations. It uses a graphical user interface to design and mix text charts with high-resolution image files. n–Title is particularly adept at building title frames, rolling credits, and still text frames, all frequently used in scientific visualization applications. It has a library of 30 high-resolution fonts which can be customized by the user, including full control over character fill, outline, and drop shadow. Text can be edited with 3-D bevels, color gradients, texture maps, and transparency. Image files in a number of popular formats can be imported to the software for use as a background or character fill. Individual n–Title frames can be sent directly to the Abekas digital video disk for inclusion in a video production.

Intentionally Left Blank

18. References

USA Ballistic Research Laboratory, Advanced Computing Systems. *The Ballistic Research Laboratory CAD Package*, Release 4.0, vols. 1–5. Aberdeen Proving Ground, MD: 1991.

Clocksin, W., and C. Mellish. *Programming in Prolog*, 3rd, rev., and extended ed. Berlin: Springer–Verlag, 1987.

Cray Research Inc. *UNICOS Primer*, SG–2010 6.0. Mendota Heights, MN: 1990.

Cray Research Inc. *UNICOS Text Editors Primer*, SG–2050. Mendota Heights, MN: 1987.

Cray Research Inc. *UNICOS Tape Subsystem User's Guide*, SG–2051 6.0. Mendota Heights, MN: 1990.

Cray Research Inc. *UNICOS CDBX Debugger User's Guide*, SG–2094 6.0. Mendota Heights, MN: 1990.

Cray Research Inc. *CF77 Compiling System, Volume 3: Vectorization Guide*, SG–3073 5.0. Mendota Heights, MN: 1991.

Cray Research Inc. *CF77 Compiling System, Volume 4: Parallel Processing Guide*, SG–3074 5.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Compiler Information File (CIF) Reference Manual*, SM–2401 1.0. Mendota Heights, MN: 1992.

Cray Research Inc. *Interlanguage Programming Conventions Technical Note*, SN–3009. Mendota Heights, MN: 1988.

Cray Research Inc. *UNICOS I/O Technical Note*, SN–3075 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Segment Loader (SEGLDR) and ld Reference Manual*, SR–0066 6.0. Mendota Heights, MN: 1990.

Cray Research Inc. *UNICOS User Commands Reference Manual*, SR–2011 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *UNICOS File Formats and Special Files Reference Manual*, SR–2014 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *UNICOS Performance Utilities Reference Manual*, SR–2040 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Cray Standard C Programmer's Reference Manual*, SR–2074 3.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Volume 1: UNICOS Fortran Library Reference Manual*, SR–2079 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Volume 2: UNICOS Standard C Library Reference Manual*, SR–2080 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *Volume 3: UNICOS Math and Scientific Library Reference Manual*, SR–2081 6.0. Mendota Heights, MN: 1991.

Cray Research Inc. *UNICOS CDBX Symbolic Debugger Reference Manual*, SR–2091 6.1. Mendota Heights, MN: 1991.

Cray Research Inc. *UNICOS X Window System Reference Manual*, SR–2101 6.0. Mendota Heights, MN: 1990.

Cray Research Inc. *CF77 Compiling System, Volume 1: Fortran Reference Manual*, SR–3071 5.0. Mendota Heights, MN: 1991.

Cray Research Inc. *CF77 Compiling System, Volume 2: Compiler Message Manual*, SR–3072 5.0. Mendota Heights, MN: 1991.

Lloyd, J. W. *Foundations of Logic Programming*, 2nd, extended ed. Berlin: Springer–Verlag, 1987.

Precision Visuals, Inc. *PicSure: user's guide*, release 2. Boulder, CO: 1985.

Stern, Nancy. *From ENIAC to UNIVAC, An Appraisal of the Eckert–Mauchly Computers.* Bedford, MA: Digital Press, 1981.

Appendix A
FY93 Charges

Intentionally Left Blank

The ARLSCF provides supercomputing time to Government activities and their contractors. In order to defray operating expenses, the following rate schedule is established for CPU use for fiscal year 1993. There is no charge for memory use, disk storage, and tape use and storage.

A.1 Hourly Usage

Hourly (pay as you go, monthly billing) customers are charged $350 per effective CPU hour. User-selected priority categories (see the chapter, "Batch Jobs") cause actual CPU hours to be weighted as follows to obtain effective CPU hours.

Nice	Category	Weight
0 – 3	Express	2.0
4 – 12	Normal	1.0
13 – 19	Deferred	0.5

A.2 Subscriptions

Subscriptions provide a means for significant users to obtain a discount, at the cost of paying in advance. There are neither refunds nor carryovers for unused portions. User-selected priority categories cause actual CPU hours to be weighted as in "Hourly Usage" to obtain effective CPU hours.

Subscription	Discounted Rate	Effective CPU Hour Allocation
$300K	$300	1000
$400K	$250	1600
$500K	$200	2500

A.3 Dedicated Time

Dedicated time is charged at $350 per wall clock hour (including transition time) times the number of CPUs on the machine. The user "owns the machine" and priorities are moot, unless the user is running multiple jobs.

A.4 Billing Questions

Questions about charges, billing, and fund transfers should be referred to the Billing Coordinator listed in Appendix B.

Intentionally Left Blank

Appendix B
Points of Contact

Intentionally Left Blank

B.1 Requests for Accounts

Users should send their requests for accounts either by electronic mail to **crayca@arl.army.mil** or by post to:

Director
U.S. Army Research Laboratory
ATTN: STEAP–IM–AC (Sharon Amerg)
Aberdeen Proving Ground, MD 21005-5067

B.2 Points of Contact

The following is a list of points of contact for various problems or questions that may arise:

Electronic mail forum	craysupport@arl.army.mil	problems, questions, discussion
Cray System Administrator	Mr. John Cole (410)278-9276 sys–admin–xmp@arl.army.mil or sys–admin–cray2@arl.army.mil	
Asst. Cray System Administrator	Mr. Mark Williams (410)278-6664 sys–admin–xmp@arl.army.mil or sys–admin–cray2@arl.army.mil	
Computer Operators	operators@arl.army.mil Cray X–MP/48: (410)278-6829 Cray–2: (410)278-6642	file restorations and tape mounts
adm.arl.army.mil System Administrator	Mr. James Fielding (410)278-6664 sys–admin–adm@arl.army.mil	
Cray Account Administrator	Ms. Sharon Amerg (410)278-6819 crayca@arl.army.mil	new accounts and clearance verification
Billing Coordinator	Ms. Judy Kelly (410)278-2820 judy@arl.army.mil	billing and fund transfer information
Secure Computer Access Team Leader	Mr. David Towson (410)278-6271 scat@arl.army.mil	
Scientific Support Team Leader	Ms. Denice Brown (410)278-6269 denice@arl.army.mil	

UNIX Support Team Leader	Ms. Lee Ann Brainard (410)278-6664 leeann@arl.army.mil
Information System Security Officer (ISSO)	Ms. Joan Ege (410)278-6265 ege@arl.army.mil
Asst. ISSO	Ms. Shirley Herbert (410)278-6276 shirl@arl.army.mil
Chief, Computational Support Branch	Ms. Linda Baldwin (410)278-7091 baldwin@arl.army.mil
Chief, High Performance Computing & Communications Branch	Mr. Robert Cahoon (410)278-6320 rmc@arl.army.mil

List of Abbreviations

AHPCRC	Army High Performance Computing Research Center
AMC	Army Materiel Command
AMSAA	Army Materiel Systems Analysis Activity
ANSI	American National Standards Institute
APG	Aberdeen Proving Ground
ARL	Army Research Laboratory
ARLSCF	Army Research Laboratory SuperComputer Facility
ARPAnet	Advanced Research Projects Agency network
AT&T	American Telephone and Telegraph
AVCO	name of company
AVS	Advanced Visual Systems (product name)
BLAS	Basic Linear Algebra Subroutines
bpi	bits per inch
BRL	Ballistic Research Laboratory
BRLESC	Ballistic Research Laboratory Electronic Scientific Computer
BRLLIB	Ballistic Research Laboratory library
BRLnet	Ballistic Research Laboratory network
BSD	Berkeley Software Distribution
BSS	a particular kind of memory allocation on Cray computers
BYU	Brigham Young University
CA	Computer Associates
CAD	Computer Aided Design
CAL	Cray Assembly Language
CDC	Control Data Corporation
CDT	Central Daylight Time
CGM	Computer Generated Metafile
CPU	Central Processing Unit
CSG	Combinatorial Solid Geometry
DDN	Defense Data Network
DEC	Digital Equipment Corporation
DISSPLA	Display Integrated Software System and Plotting LAnguage
DoD	Department of Defense
DSN	Defense Switched Network
DSNET1	Defense Secure Network 1
EA	extended addressing, a feature of certain Cray computers
EDT	Eastern Daylight Time
EDVAC	Electronic Discrete Variable Automatic Computer, an early computer
EISPACK	Eigensystem Package
EMA	extended memory addressing, a feature of certain Cray computers
EOF	end of file
EST	Eastern Standard Time
FTP	File Transfer Protocol
GB	gigabyte
GKS	Graphical Kernel System
GUI	Graphical User Interface
HDF	Hierarchical Data Format
HP	Hewlett Packard
HULL	name of computer code; no meaning

IBM	International Business Machines
id	identification, as in "id number"
IEEE	Institute of Electrical and Electronics Engineers
IMSL	International Mathematical and Statistical Libraries
I/O	input/output
IPI	Intelligent Peripheral Interface
ISSO	Information System Security Officer
Kbyte	kilobyte
LAN	local area network
LINDO	Linear, INteractive, Discrete Optimizer
LINPACK	Fortran subroutine package for solving systems of linear equations
MB	megabyte
MGED	Multidevice Graphics EDitor
MILnet	Military network
MIL–STD	Military Standard
MIT	Massachusetts Institute of Technology
MPGS	MultiPurpose Graphics System (Cray software)
MSC	MacNeal–Schwendler Corporation
MW	megaword
NASA	National Aeronautics and Space Administration
NIST	National Institute of Science and Technology
NQS	Network Queuing System
NSFnet	National Science Foundation network
NTSC	National Television Systems Committee
PACX	Public Access Computer eXchange
PC	IBM's version of personal computer, or one compatible therewith
PVI	Precision Visuals, Inc.
RDEC	Research, Development, and Engineering Center
RGB	red, green, blue
SCSI	Small Computer Serial Interface
SGI	Silicon Graphics, Inc.
SOP	Standard Operating Procedure
SP	Standard Play (VHS usage) or Superior Performance (U-matic and Betacam usage)
stderr	standard error
stdin	standard input
stdout	standard output
TAC	Terminal Access Controller
TCP/IP	Transmission Control Protocol/Internet Protocol
VAX	Virtual Address eXtension, a family of DEC computers
VCR	Video Cassette Recorder
VTR	Video Tape Recorder

No. of Copies	Organization
2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22304-6145
1	Commander U.S. Army Materiel Command ATTN: AMCAM 5001 Eisenhower Ave. Alexandria, VA 22333-0001
1	Director U.S. Army Research Laboratory ATTN: AMSRL-OP-CI-AD, Tech Publishing 2800 Powder Mill Rd. Adelphi, MD 20783-1145
1	Director U.S. Army Research Laboratory ATTN: AMSRL-OP-CI-AD, Records Management 2800 Powder Mill Rd. Adelphi, MD 20783-1145
2	Commander U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-IMI-I Picatinny Arsenal, NJ 07806-5000
2	Commander U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-CCB-TL Watervliet, NY 12189-4050
(Unclass. only) 1	Commander U.S. Army Rock Island Arsenal ATTN: SMCRI-IMC-RT/Technical Library Rock Island, IL 61299-5000
1	Director U.S. Army Aviation Research and Technology Activity ATTN: SAVRT-R (Library) M/S 219-3 Ames Research Center Moffett Field, CA 94035-1000

No. of Copies	Organization
1	Commander U.S. Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010
1	Commander U.S. Army Tank-Automotive Command ATTN: ASQNC-TAC-DIT (Technical Information Center) Warren, MI 48397-5000
1	Director U.S. Army TRADOC Analysis Command ATTN: ATRC-WSR White Sands Missile Range, NM 88002-5502
1	Commandant U.S. Army Field Artillery School ATTN: ATSF-CSI Ft. Sill, OK 73503-5000
(Class. only) 1	Commandant U.S. Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only) 1	Commandant U.S. Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905-5660
1	WL/MNOI Eglin AFB, FL 32542-5000
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen
1	Cdr, USATECOM ATTN: AMSTE-TC
1	Dir, ERDEC ATTN: SCBRD-RT
1	Cdr, CBDA ATTN: AMSCB-CI
1	Dir, USARL ATTN: AMSRL-SL-I
10	Dir, USARL ATTN: AMSRL-OP-CI-B (Tech Lib)

Intentionally Left Blank

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-TR-150 Date of Report June 1993

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

BUSINESS REPLY MAIL
FIRST CLASS PERMIT No 0001, APG, MD

Postage will be paid by addressee.

Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-CI-B (Tech Lib)
Aberdeen Proving Ground, MD 21005-5066

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES