

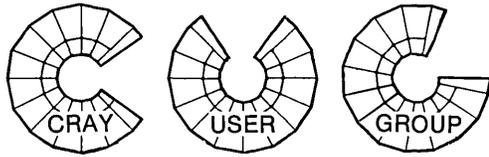


INCORPORATED



PROCEEDINGS

FALL 1985



INCORPORATED

PROCEEDINGS

SIXTEENTH SEMI-ANNUAL CRAY USER GROUP MEETING

September 30 - October 3, 1985

Hôtel du Parc
Montréal, Québec
Canada

Host: Environnement Canada

Karen Friedman, Editor

Prepared for publication and printed at the National Center for Atmospheric Research, Boulder, Colorado (NCAR).*,†

* The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation.

† Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

CONTENTS

PRELIMINARY INFORMATION

Acknowledgements.	3
CRAY User Group, Inc. Board of Directors, 1985-1986	4
Members of the Program Committee.	5
Program	6

PRESENTATIONS

Cray Corporate Report, Robert H. Ewald.	9
CRAY Software Status, Margaret A. Loftus.	11
CFT77: CRAY's New Fortran Compiler, Karen Spackman	13
Multitasking in Operating Systems, Jeffrey C. Huskamp . . .	15
CFT Compiler Optimization and CRAY X-MP Vector Performance, Ingrid Y. Bucher, Margaret L. Simmons	19
NAS Kernel Benchmark Results, David H. Bailey	22
SSD User Experience Session, Mostyn Lewis	26
CRAY SSD Scheduling Concerns, Ronald Kerry.	27
SSD Experience at Boeing Computer Services, Conrad Kimball.	28
User Requirements Committee Report, Stephen Niver	33

SHORT PAPERS

Some User Experience in Migrating to CFT1.14, Chris Lazou .	39
---	----

SPECIAL INTEREST COMMITTEE REPORTS

Networking and Frontends Session I, Dean W. Smith	43
CRAY Integrated Support Processor Installation Experience, Ronald Kerry.	43
Running Multiple CRAY Stations at Chevron Oil Field Research Co., Annabella Deck.	44
Enhanced Station Messages Support, Dean Smith	45
Languages Session, Mary Zosel	47
CFT77 Discussion, Karen Spackman.	48
Special Interest Group on Operations, Gary Jensen	50
Computer Operations at Environment Canada, Gary Cross . .	51
FOCUS at the Los Alamos National Laboratory, Fred Montoya.	56
Multitasking Performance Workshop Summary, Ann Cowley . . .	59
Multitasking the Weather, David Dent.	60
CMTS - A CRAY Multitasking Simulator, J.D.A. David.	68
Multitasking, Margaret L. Simmons	72
The Multi-Level Data Bus Approach to Multitasking, J.L. Owens.	72
Experiences with CRAY Multitasking, Eugene N. Miya, M.S. Johnson.	77

CONTENTS

SPECIAL INTEREST COMMITTEE REPORTS (cont.)

Multitasking, Margaret L. Simmons (cont.)	
Speedup Predictions for Large Scientific Parallel Programs on CRAY X-MP-Like Architectures, Elizabeth Williams, Frank Bobrowicz	100
Los Alamos National Laboratory Control Library, F.W. Bobrowicz.	101
Networking and Frontends Session II, Dean W. Smith.	102
MVS Station Status, Peter Griffiths	102
Superlink Status, Stuart Ross	102
Apollo and VAX Station Status, Martin Cutts	103
VM and UNIX Stations, Allen Machinski	103
CRAY CYBER Link Software: Product Status and Development Plan, Wolfgang G. Kroj.	104
COS Session, David Lexton	106
COS Experience Panel.	106
Claus Hilberg	106
Conrad Kimball.	107
Mostyn Lewis.	108
COS Interactive: A Developer's Perspective, Bryan Koch .	109
OSSIC Report, David Lexton.	114
Microtasking Panel Session, Mary Zosel.	115
Performance and Evaluation Special Interest Committee, Mostyn Lewis.	116
I/O Workshop, Mostyn Lewis.	117
Benchmarking CRAY's X-MP and SSD, Christopher Holl	118
Software Paging for Large Matrices on CRAY X-MP, U. Detert	135
New COS Features, Clay Kirkland	145
Customer Experience with DD-49 Disks, Mostyn Lewis. . . .	158
Report on the Graphics Session, H.E. Kulsrud.	159

ADDITIONAL REPORTS

President's Report, M.G. Schomberg.	163
Incorporation of the CRAY User Group, M.G. Schomberg. . . .	165
Report of the Vice President, H.E. Kulsrud.	166
Report of the Program Committee, David Lexton	167

ADDITIONAL INFORMATION

Montreal CUG Meeting Participants by Organization	171
CUG Site Contact List	180
Call For Papers Form.	203
Help CUG Form	205

PRELIMINARY INFORMATION

ACKNOWLEDGEMENTS

Thanks go to the following persons who comprised the Local Arrangements Committee from Environment Canada:

- * Raymond Benoit - Chair
- * Betty Benwell
- * Gary Cross
- * Jean-François Gagnon
- * Claude Payette

Thanks also go to Kathy Lucero and Mary Buck for their assistance in typing various contributions to these Proceedings.

CRAY USER GROUP, INCORPORATED

BOARD OF DIRECTORS

1985 - 1986

<u>TITLE</u>	<u>NAME</u>	<u>ORGANIZATION</u>
President	Helene Kulsrud	IDA
Vice President	David Lexton	ULCC
Treasurer	Robert Price	Westinghouse
Secretary	Karen Friedman	NCAR
Member, Board of Directors	Stephen Niver	BCS
Member, Board of Directors	Sven Sandin	SAAB-Scania AB
Member, Board of Directors	Michael Schomberg	AERE-Harwell

Members of the Program Committee

Raymond Benoit	-	EC
Ann Cowley	-	NCAR
Jacqueline Goirand	-	CIFRAM
Gary Jensen	-	NCAR
Helene Kulsrud	-	IDA, Chair
Mostyn Lewis	-	CHEVRON
David Lexton	-	ULCC
Jerry Melendez	-	LANL
David Sadler	-	CRI
Margaret Simmons	-	LANL
Dean Smith	-	ARCO
Mary Zosel	-	LLNL

PROGRAM
CRAY USER GROUP, INC.
FALL 1985
MONTREAL

TUESDAY OCTOBER 1		WEDNESDAY OCTOBER 2		THURSDAY OCTOBER 3	
8:30	Welcome B. Attfield (EC)		Multitasking Performance I A. Cowley (NCAR)		I/O M. Lewis (Chevron)
8:45	Keynote J. Connolly (NSF)				
9:15	CRI Corporate Report B. Kasson (CRI)	8:30	Operations II G. Jensen (NCAR)	8:30	
9:30	CUG Report M. Schomberg (AERE)				Graphics H. Kulsrud (IDA)
9:45	CUG Elections J. Goirand (CISI)		CTSS G. Melendez (LANL)		
10:00 BREAK		10:00 BREAK		10:00 BREAK	
10:30	Presentation of New Officers	10:30	Multitasking In Operating Systems J. Huskamp (IDA)	10:30	Effects of Compiler Optimization on X-MP I. Bucher (LANL)
10:35	CRI Software Report M. Loftus (CRI)	11:00	Microtasking Overview M. Booth (CRI)	11:00	NAS Kernel Benchmark Results D. Bailey (NASA/Ames)
11:00	CFT-77 K. Spackman (CRI)	11:30	Synchronization Speed and Multiprocessor Performance T. Axelrod (LLNL)	11:30	Microtasking Benchmarks at CRI L. Kraus (CRI)
11:30	CRI Directions In Networking D. Thompson (CRI)				
12:00 LUNCH		12:00 LUNCH		12:00 LUNCH	
	Networking D. Smith (ARCO)		Multitasking Performance II M. Simmons (LANL)	1:30	User Requirements Report S. Niver (Boeing)
1:30		1:30	Front Ends D. Smith (ARCO)	1:45	SSD Update Users Report M. Lewis (Chevron)
	Multitasking Tutorial R. Brickner (LANL)		Short Papers J. Goirand (CISI)	2:00	CRAY II Performance R. Numrich (CRI)
3:00 BREAK		3:00 BREAK		2:45	CRAY-2 Users Report J. Perdue (NASA/Ames)
	Languages M. Zosel (LLNL)		COS D. Lexton (ULCC)	3:15	Next Conference S. Niver (Boeing)
3:30		3:30	Microtasking Panel M. Zosel (LLNL)	3:25	CLOSING REMARKS
	Operations I G. Jensen (NCAR)			3:30 BREAK/END	
7:00 CONFERENCE DINNER		5:00 PROGRAM COMMITTEE		4:00	CUG Advisory Council
				4:30	User Requirements Committee

PRESENTATIONS

Cray Corporate Report

Robert H. Ewald

Cray Research, Inc.
Minneapolis, MN

The first three quarters of 1985 have been very busy and productive for Cray Research, Inc. The sections below briefly review Cray's organization, business, and product highlights during 1985.

We expect to install about 30 new systems and to reinstall several other systems during 1985. To date, the systems shown in Table 2 have been installed during 1985.

ORGANIZATION

Cray continues to operate in a decentralized manner believing that small groups of people dedicated to a common task with limited resources and aggressive implementation schedules work well in the fast moving high performance computing business. As of September 30, 1985, Cray had about 3,000 employees distributed in the following major functional areas:

- 64% - Hardware Dev., Eng., & Mfg.
- 22% - Field Support and Marketing
- 10% - Software
- 4% - Finance, Human Resources, and Admin.

BUSINESS

Cray's 1985 business continues to be very strong. We hope to receive at least 35 orders for our systems during 1985. To date we have received 28 orders including those indicated in Table 1.

Table 1
1985 Orders Thru 9-30-85

<u>North America</u>		<u>International</u>	
LLNL-LCC	X-MP/48	ADNOC	X-MP/14
SNLA	X-MP/24	ECMWF	X-MP/48
Bell	X-MP/24	EDF	X-MP/216
U of MN	CRAY-2	ELF	X-MP/12
Chevron, TX	X-MP/24	MPPPI	X-MP/24
NASA/Lewis	X-MP/24	RUS	CRAY-2
*ORNL	X-MP/12	*EPFL	CRAY-1/S2000
*Shell	X-MP/14	*Nissan	X-MP/11
*DuPont	CRAY-1/A	*BP	X-MP/12
*GA Tech.	X-MP/48		
*U of IL	X-MP/48		
*GD, TX	X-MP/24		
*NASA/Marshall	X-MP/48		
*Fairchild, CA	CRAY-1S2000		
*Lockheed, CA	X-MP/24		
*Wright-Patt.	X-MP/12		

Table 2
1985 Installations Thru 9-30-85

<u>North America</u>			
BCS	X-MP/24	*Ford	X-MP/11
Exxon USA	X-MP/14	*NRL	X-MP/14
Chevron, CA	X-MP/48	*Rockwell	X-MP/14
LANL	X-MP/48	*Northrop	X-MP/12
LLNL-LCC	X-MP/48	*ORNL	CRAY-1/S200
SNLA	X-MP/24	*Chevron, TX	X-MP/24
		*Lockheed, CA	X-MP/24
		*AFGWC	X-MP/22
		*GD	X-MP/24
		*U of IL	X-MP/24
		*Wright-Patt.	X-MP/12

International

*CINECA	X-MP/22
*Toshiba	X-MP/22
*Opel	CRAY-1/S1000

PRODUCT HIGHLIGHTS

Two major product announcements were made during the summer of 1985. The CRAY-2 was formally announced in June 1985 as a four processor parallel vector computer with a 4.1 ns cycle time and 256Mw of memory.

The first CRAY-2 was installed at the Magnetic Fusion Energy Computer Center in Livermore, CA in May 1985. The second system was shipped to NASA's AMES Research Center in September 1985.

In September the X-MP family was expanded with larger memory (up to 16Mw) systems. The gather/scatter, compress index and a second vector logical unit were also implemented across the X-MP line. The current X-MP family of systems is shown in Table 3.

* New Accounts

Table 3

X-MP Systems

<u>MODELS</u>	<u>NO. of CPUs</u>	<u>MEMORY</u>		<u>No. of COLs</u>	<u>No. of IOPs</u>	<u>S&D SIZE (MW)</u>
		<u>SIZE (MW), TYPE</u>	<u>NO. of BANKS</u>			
X-MP/11	1	1, MOS,16	6	2-4	32-128	
X-MP/12	1	2, MOS,16	6	2-4	32-128	
X-MP/14	1	4, MOS,16	6	2-4	32-128	
X-MP/18	1	8, MOS,32	6	2-4	32-128	
X-MP/24	2	4, MOS,16	8	2-4	32-128	
X-MP/28	2	8, MOS,32	8	2-4	32-128	
X-MP/216	2	16, MOS,32	8	2-4	32-128	
X-MP/48	4	8, ECL, 32	12	4	32-128	
X-MP/416	4	16, ECL, 64	12	4	32-128	

SUMMARY

The first three quarters of 1985 have been very productive for Cray Research with announcements and installations of new systems. The demand for CRAY systems continues to be very strong as new scientific, engineering and national defense applications are developing and as existing applications continue to grow.

CRAY SOFTWARE STATUS

Margaret A. Loftus

Cray Research, Inc.
Minneapolis, MN

The following software has been released since the last User Meeting five months ago.

1.14 Nos Station	May 1985
1.14 On-Line Diagnostics	August 1985
1.0 VAX/UNIX Station	September 1985
1.14 NOS/BE	September 1985
1.14 COS/CFT BF3	September 1985

The last major software release was COS/CFT 1.14 in January 1985.

The field have experienced very few problems with the COS 1.14, the most significant being Cyber tape related. Some sites have encountered stability problems with CFT 1.14. As a result of these problems, we will be making changes in CFT testing.

The following is a status of the major software plans in progress

- 1.15 will be released late 85/early 86.
- CAL 2.0 to be released the end of 1985 and will support the CRAY-1, CRAY-1S, CRAY-1M, CRAY X-MP and CRAY-2.
- Pascal 3.0 to be released the end of 1985. Performance improvements include scalar, vectorization via array syntax and multi-tasking.
- C under COS to also be released by the end of 1985. A prerelease is available today.
- CFT77 (NFT) is being tested in Mendota Heights and planned to be released by the end of 1985/early 1986. The first release will support the X-MP and the second release the CRAY-2. Results thus far have been excellent in both performance and stability.
- Cyber Station development moved to Germany. 1.15 NOS Station (dual state) to be released 4Q85. 1.15 NOS/BE Station (dual state) 4Q85 release dependent CDC release. 1.16 NOS Station release 2Q86 to include interactive. The NOS/VE Station effort has begun and targeted for 1987 release.
- Superlink/MVS. Superlink/ISP installed at a customer site in October. The Superlink/MVS R2 to be released in 1986 and provide interactive support. Superlink/MVS R3 to be released in 1987 and provide full Station support and full intergration into SNA.
- 2.0 MVS Station (1.14) released 1Q86 and include interactive.
- 3.0 VM Station to be released in November and include dispose to mini disk; separate data streams for graphics; 3375 disk support and improved tape handling.
- 3.0 VAX/VMS released October with enhanced DECnet support; VMS 4.0 is required.
- 2.0 Apollo Station release 1Q86 with operator commands and control.
- Microtasking for the X-MP will be included in 1.15. A prerelease will be available 1Q86 under COS.
- CFT2 - CRAY 2 Fortran compiler is based on CFT. Development is complete and we are now stabilizing.
- Multitasking - Future microtasking support will differ from the X-MP due to hardware differences.
- CX-OS (UNIX*) supports CRAY-1s, CRAY X-MP and CRAY-2. The initial release in 1Q86 will include CFT, CFT2, CFT77, C, Pascal, Segldr, multitasking (macro/micro on X-MP). Some tape support on X-MP (no label processing), batch, debuggers, source code maintenance, SCILIB, X-MP and CRAY-2 peripherals, large memory support, I/O enhancements (striping, device overflow, performance, streaming large files) TCP/IP (separate package available from third party) and interactive.

We expect to pass AT&T's System V Unix validation tests when they are available.

*UNIX is a trademark of AT&T Bell Laboratories.

A subsequent release is planned for 4086 with SCP (Station) recovery. In 1988 we expect to have a common CX-OS software product across all hardware products with equivalent functionality to COS at that time.

Major emphasis of CRI software over the next three years are in the following four areas:

- Multitasking
- High Performance Transportable Software to deal with future hardware architectures
- Connectivity which includes enhancements to existing and new stations, and networking.
- Continued Enhancements and Support of existing software.

CFT77: CRAY'S NEW FORTRAN COMPILER

Karen Spackman

Cray Research, Inc.
Mendota Heights, Minnesota

Cray will be releasing a new FORTRAN compiler for its machines early in 1986. The first release will be for the CRAY X-MP under COS. Subsequent releases will support the CRAY-2, the CRAY X-MP under UNICOS, and the CRAY-1. We have been working on this project for some time, and many of you have heard it referred to by its working name of NFT. During 1985 the name CFT77 was selected for the release name of the compiler. When we began this project four years ago, we had several goals we wanted to achieve. I will review four of the most important of these goals.

One of our primary goals was to produce a compiler that could be ported more easily to new CRAY machines. At the time we started the project, the CRAY X-MP and CRAY-2 had not yet been announced, but we knew not only that these machines were coming shortly but that there would be follow-on machines in both lines. All of these machines have to be supported as quickly as possible with a high-quality FORTRAN compiler that generates high-performance code.

What do we mean by portability? The definition that we use states that portability is a measure of the ease with which a program can be moved from one environment to another without losing the characteristics that made moving the program desirable. For us, our FORTRAN compilers must produce high-performance generated code. If we lose that performance when we port the compiler, then our compiler was not very portable either.

Compiler portability may seem to be an issue only for the implementor, but it has an important implication for the user as well. We want the user's view of FORTRAN to be the same across our machine lines; we want the same FORTRAN "dialect" to be supported on all the machines. This can best be achieved by having the same compiler on all machines.

A second goal for the new compiler was to incorporate some of the work in optimization that has been done in the last several years that we could not reasonably put into CFT. In particular we expected this to give us improved scalar performance.

Our third goal developed out of the realization that with our new machines we were moving into multiprocessors and that we would need to make utilizing the multiple processors as easy as possible for our users. Consequently one of the goals for the new compiler was to develop a vehicle that could be used to partition codes automatically for multitasking. Because of the extensive analysis that will be needed to do this, CFT is not an appropriate vehicle. Automatic multitasking will not be available in the first release of CFT77, but we expect to demonstrate the feature by the end of 1986.

Finally, we wanted to develop a basis for future compiler development work for other languages. Since we are making a considerable investment in optimization and since we are developing code generators for all of our mainframes, we wanted to take advantage of this work for additional languages. In 1981 FORTRAN was the only language we supported; since then we have added Pascal and C as supported languages and expect to offer additional languages in the future. We plan to develop a common modular compiling system out of the CFT77 development work and use this to implement FORTRAN 8X as well as new C and Pascal compilers.

One requirement for the new compiler from the beginning was that it be compatible with CFT. There are two parts to this requirement. One is FORTRAN source compatibility: a FORTRAN program that works correctly with CFT should work with CFT77. To this end, CFT77 supports the existing CFT extensions with few exceptions. The second part of this requirement is relocatable compatibility: routines compiled by CFT77 can call and be called by routines compiled by CFT (with the new calling sequence), and CFT77 uses the same run-time libraries as CFT.

There are certain differences between the compilers that the user will notice. CFT77 does more extensive error checking than CFT, and some constructs that may appear to work with CFT will be detected as errors. We are trying to identify these and will provide a list of differences with the release information. With CFT77 we have made POINTER a

separate data type and placed some restrictions on the operations that can be performed on pointers. We think this will give the user some additional protection from inadvertently misusing pointers without invalidating existing programs. If we find that this change does invalidate existing programs, we will change to the CFT implementation. We expect to find this out during beta testing since one of the beta test sites uses pointers extensively. Finally I want to point out that CFT77 and CFT are different compilers and have different control card options and compiler directives. However, since compiler directives are embedded in the source code, CFT77 recognizes CFT directives that it doesn't use and prints warning messages indicating that the directives are ignored.

CFT77 is a FORTRAN 77 compiler. We have also added three features that we expect to be included in the next standard. We allow internal identifier names up to 31 characters in length, and these may include underscores. We have implemented a subset of the array syntax proposed for FORTRAN 8X. This allows the user to write expressions involving whole arrays or sections of arrays; for the first release we limit the use to assignment statements and do not allow character type. We have also implemented automatic arrays which allow an adjustable array to be allocated on entry to a subroutine and freed on exit.

The approach to optimization used in CFT77 is different from that used by CFT. CFT77 does what is termed global optimization which means that an entire subprogram is analyzed to determine the control and data flow. Scalar optimizations are then done using the flow information; this means that optimizations are not applied only within basic blocks as they are with CFT. Further, the information gathered for scalar optimization is used in the vectorization analysis and in doing register assignment.

Future development areas for CFT77 include extending the vectorization facility and adding an assertion capability to let the programmer give the compiler information to aid vectorization. Automatic partitioning for multitasking is another area for continued development work; we expect this capability to

develop and be extended over several releases as we learn more about how we can profitably use multiple processors. We plan to extend the array syntax subset that we have implemented in the first release. And, of course, we expect to make performance improvements continually as we evaluate the code we produce and identify areas to improve.

¹ Lecarme, Olicier, and Peyrolle-Thomas, Marie-Claude, "Self-compiling Compilers: An Appraisal of Their Implementation and Portability", Software Practice and Experience, 8, 149-170 (1978).

Multitasking in Operating Systems

Jeffrey C. Huskamp

Institute for Defense Analyses
Princeton, NJ

Abstract

There are several implementation problems in bootstrapping an existing operating system, which is written for a single processor machine, to a multiprocessor machine with a large memory. The problems include deciding whether to make the kernel multithreaded, implementing multiple execution points (tasks) in a process, avoiding deadlock and minimizing context switching overhead. Some possible solutions for CRAY machines are discussed.

Key words: multiprocessing, multitasking, CRAY, supercomputer

Introduction

All four production operating systems for CRAY machines (COS[1], CTSS[2], Folklore[3], AMOK[4]) and the announced operating system for the CRAY-2 (UNIXTM [5]) have ancestors that only execute on single processor machines. The next high performance supercomputers will provide multiple processors to speed up program execution. Since designing, implementing, and changing over to an entirely new operating system that makes use of multiple processors is a very laborious undertaking, some of these operating systems will probably be modified to permit the users access to the multiprocessing/parallel processing features of the machines. However, incorporating multiple processors into the user interface and making use of the multiple processors inside the operating system cannot be done easily. Some of the features needed in multiprocessor operating systems that are not found in single processor systems include permitting the operating system to have multiple execution threads active at one time, permitting user jobs to have more than one execution point (i.e. multiple tasks), preventing system deadlock, and avoiding full context switches when possible since the process state information is large and getting larger.

This paper discusses different approaches that can be taken toward solving these problems. For a complete solution, changing a majority of the operating system may be necessary. When possible, a low cost partial solution is identified that may make a full solution unnecessary.

Single Threaded Kernels

Most operating system kernels assume that only one processor executes in the operating system at one time. In bootstrapping an operating system to a multiprocessor machine, some consideration might be given to utilizing multiple processors within the operating system. There are at least three approaches to this problem. The easiest approach is to lock the entire operating system so only one processor is executing the operating system at one time. With this strategy, one processor could be waiting for another processor to finish its tour through the system. If this condition occurs frequently, processors are waiting for system calls to complete and the operating system becomes a performance bottleneck. To obtain an estimate of the performance degradation, a quick calculation (with many independence assumptions) can estimate the probability that two or more processors could be executing system calls at the same time. For the calculation, the percentage of CPU time devoted to executing system calls in AMOK (=5%) will be used as an example. Assuming that the system has N processors, the probability of conflict is:

N= number of processors

S= probability of wanting to execute a system call=0.05

P= probability 2 or more processes are executing system calls

P= 1-(probability that 0 or 1 processors are executing system calls)

$P = 1 - ((1-S)^N + NS[(1-S)^{(N-1)}])$

# Processors	Conflict Probability
1	0.0000
2	0.0025
3	0.0073
4	0.0140
5	0.0226
6	0.0328
7	0.0444
8	0.0572
16	0.1892
32	0.4800

The above table suggests that for machines having four or fewer processors, the global operating system locking approach seems to not degrade performance significantly. This is consistent with observations of CTSS on multiprocessor machines. However, for the next generation of supercomputers that have a larger number of processors, this approach does not seem to be correct.

The next simplest possibility for taking advantage of multiple processors is to provide a shared kernel that performs the basic I/O and context switching functions, and supports multiple operating systems. This breaks the host multi-processor machine into multiple guest systems with a small number of processors in each system. If enough guest systems are introduced, the argument for locking the entire kernel that was made above may again be valid. This approach can be thought of as a virtual machine implementation. The advantage is that different operating systems can execute at the same time (e.g. a batch system and a timesharing system) to provide different services. The disadvantages are that (1) more memory is devoted to the operating system, (2) the lower level kernel can be a bottleneck unless it is reentrant, (3) an extra context switch is needed to access the shared kernel since each guest operating system must be treated with suspicion, (4) extra checking must be included in the shared kernel so system bugs from one guest system don't crash another guest system, and (5) all system resources (e.g. memory, disk) must be partitioned. The peripheral partitioning also may imply that multiple copies of public files are necessary, one for each guest system. An example of this approach is the NLTSS [6] development system at Livermore. In this case, one operating system is a production system (LTSS) and the other is a development system (NLTSS).

Finally, the most expensive approach is to actually redesign the operating system to take advantage of multiple processors. This involves setting up tasks within the system that have their own stacks, developing a low cost system task switching mechanism, and locking all shared

data structures. This approach could result in less efficiently compiled code for the operating system and would create synchronization overhead for processors executing inside the system.

One Execution Point per Process

Perhaps the most troublesome problem in providing support for parallel processing is permitting multiple exchange packages inside the process header (minus words). The first concern is the expansion of the header by at least the amount of storage that constitutes a process context. In AMOK on the CRAY-1, this amounts to a minimum of 657 words per task. In addition, descriptions of open objects for each task consume more space. On the CRAY-2, the 16K local memory adds more storage overhead to tasks.

With one task per process, all execution in the process stops when a system call is issued. That is, the user exchange package is waiting for the system call processing to complete. In parallel processing, other tasks in the process may be active and can cause system call implementations that work in single processor systems to fail. For example, trying to terminate all tasks within a process can be adventuresome since some tasks may be issuing system calls that take a long time to complete or may be erroneously spawning other tasks. Certainly logic must be added to make sure all tasks are moving toward termination at the same time.

The system call interface must be expanded to enable task creation and task deletion. Other system calls, such as suspend process, must be extended to include tasks. This also implies that the scheduler must be modified to service these tasks.

If these changes are troublesome, perhaps an implementation which pushes some of the implementation into the user space would be better. For example, the CTSS approach puts task creation and deletion into user library routines [7].

Another alternative is to adapt a UNIX-like system that thrives on many small processes but does not support multiple execution points within one process. Allocating multiple CPU's in this situation is straightforward. However, to avoid idle CPU's, the number of jobs in the queue must at least equal the number of physical CPU's. This will make the throughput of the machine look good, but will not help the response time of any one job. Of course, if extensions are added to UNIX to permit multiple execution points within one address space, a new version of UNIX

will be required. This may not be the correct path to take since compatibility problems may be created with other versions of UNIX.

Errors in Multithreaded Systems

One of the most feared problems in multi-processor systems is system deadlock. The problem occurs when two or more processors that are executing inside the operating system try to reserve the same objects but in a different order. To eliminate this problem, all execution paths of the operating system must be checked for the possibility of multiple object reservations, which is a time-consuming procedure. One fact that helps the problem is that the large majority of system calls usually only needs to reserve one object at a time. Thus many execution sequences do not need to be analyzed. For example, directory calls such as create, open and delete only need to reserve the directory to be modified. In AMOK, system calls such as initiate process and terminate process have the potential to reserve more than one process so these execution paths must be scrutinized.

To reduce the number of system tables and/or system table entries that must be locked, some of the operating system can be structured so that only one processor executes within a certain subsystem at a time. Three subsystems that might be handled this way are the memory manager, the network manager and the operator console manager. Since these subsystems are most easily programmed as single processor tasks and are crucial for correct system performance, locking these subsystems at a very high level seems acceptable. This utilizes the message-system approach to structuring an operating system [8] as opposed to the procedure-oriented approach. Thus there appear to be some operating system functions that can have a very coarse grain of protection (and can be more easily programmed) and some that must have very fine-grained protection.

Context Switch Time

For jobs that request a large number of system functions per CPU second, context switching can represent a significant amount of overhead. This is particularly true if the B, T and V registers must be saved. Two mechanisms contained in current operating systems can help reduce this overhead. The simplest mechanism permits more than one system call to be issued with one context switch to the operating system. One implementation is done by CTSS which allows system calls to be chained together. For example, this can

speed up tape processing by reading or writing multiple records per context switch. On AMOK, some non-standard arithmetic/logical and conditional branching system calls have been implemented so that simple functions can be computed without exchanging back to the user process. This is useful in loop control and in termination condition testing. This makes system calls resemble assembly language programs. The system calls take the same amount of time as a round-trip context switch from the user process to the operating system and back. Thus system call programs with simple loop control can execute faster than issuing single system calls. This optimization is widely used by user support routines as evidenced by the system-wide average of four system calls executed per exchange to the system. Unfortunately, the standard UNIX system does not permit more than one system call to be issued per context switch. It would also seem unwise to modify this basic part of the UNIX user interface in attempts to speed up the system.

The more complex mechanism is the implementation of lightweight tasks within a process by code executing within the process. For user level lightweight tasks, the operating system does not know there are multiple tasks in the process. This means that task scheduling, task dispatching and context switching are done by the process itself. Lightweight tasks have been implemented for NLTSS servers and for the AMOK operating system. As an example of lightweight tasks, the AMOK implementation will be discussed in slightly more detail.

When an AMOK system task wants to temporarily give up control of the CPU, it calls the procedure STKSLEEP, which saves the contents of register B02 in variable STKINFO for the task. In the CRAY calling sequence, B02 points to the register save area for the procedure that called STKSLEEP. The scheduler task is then awakened to dispatch another task. To awaken a task, STKWAKE restores B77, B00, B01, B02, B66 and B67 from the register save area pointed to by STKINFO for the task. This restores the registers to the correct state for returning to the procedure that called STKSLEEP. The transfer of control is accomplished by a non-local jump (not a normal procedure return) implemented in assembly language. The procedure that called STKSLEEP thinks that a normal return from STKSLEEP has occurred and resumes its execution.

The advantage of lightweight tasks is that context switching incurs only a small overhead. Not all the registers need to

be saved and, for user level tasks, an exchange to the operating system for scheduling is not needed. The disadvantage is that an infinite loop in one task may disable the entire process.

Summary

The four problem areas discussed in this paper are prime areas of concern when existing operating systems for single processor machines are bootstrapped to multiprocessor machines. The approaches suggested here are extensions of current efforts being made to attack these problems. Hopefully some of these concerns will be addressed in the Unix implementation underway at CRI.

REFERENCES:

- [1] Cray Research, Inc., "CRAY-OS (COS) Version 1 Reference Manual", SR-0011, Revision N, (to appear).
- [2] Fong, K., "The NMFECC Cray Time-Sharing System", Software Practice and Experience, 15 (1), 87-103 (January 1985).
- [3] Santavicca, T., "Folklore - Delivering Supercomputing", Proceedings of the Conference on Operating Systems for Supercomputers, Princeton, New Jersey (June 1984).
- [4] Huskamp, J.C., "A Modular Operating System for the CRAY-1", (to appear).
- [5] Ritchie, D.M. and K. Thompson, "The UNIX Time-Sharing System", Comm. Assoc. Comp. Mach., 17 (7), 365-375 (July 1974).
- [6] Donnelley, J., "Components of a Network Operating System", Computer Networks, 3, 389-399 (1979).
- [7] Long, G., "Status of CRAY-2 Software at MFE", Proceedings of the Conference on Operating Systems for Supercomputers, Princeton, New Jersey, (June 1984).
- [8] Lauer, H., "On the Duality of Operating Systems Structures", in Proc. Second International Symposium on Operating Systems, IRIA, October 1978, reprinted in Operating Systems Review, 13 (2), 3-19 (April 1979).

CFT COMPILER OPTIMIZATION AND CRAY X-MP VECTOR PERFORMANCE

Ingrid Y. Bucher and Margaret L. Simmons

Los Alamos National Laboratory
Los Alamos, New Mexico

INTRODUCTION

The motivation for this paper was the observation that Cray X-MP vector performance has not always evolved in the expected direction of improvement. It is well known that vector performance of register-to-register vector computers is strongly compiler dependent. The compiler is responsible for proper sequencing of vector load, arithmetic, and store operations, and for scheduling necessary scalar operations in such a way that they take place while vector operations are in progress and therefore do not consume any additional time. We have analyzed vector performance data for CFT compiler versions 1.10 through 1.14. It is concluded that in spite of the great performance improvements achieved by version 1.14 of CFT, further speedups are possible by eliminating the slowdowns introduced in version 1.11.

MODEL OF CRAY VECTOR PERFORMANCE

Based on the well known fact that Cray vector operations are "stripmined" in sections of 64, the time required to perform arithmetic operations on vectors of length N is given by

$$T = T_{startout} + \left\lceil \frac{N}{64} \right\rceil * T_{startstrip} + N * T_{element} \quad (1)$$

where the brackets $\left\lceil \frac{N}{64} \right\rceil$ denote the next integer greater than or equal to $N/64$, and where $T_{startout}$ is the startup time for the outer loop processing the strips, $T_{startstrip}$ the startup time for each strip of length 64, and $T_{element}$ the time required to produce one result element. Equation (1) represents a linear step function as represented in Fig. 1, with the height of each step equal to the startup time of each 64-strip. In reality, there is a small overshoot associated with the startup of each 64-strip (see Fig. 2). This is due to the fact that for vector instructions with very short vector lengths some necessary scalar work is incompletely overlapped.

The points for N equal to multiples of 64 lie on a straight line represented by

$$T = T_{startout} + N * (T_{startstrip}/64 + T_{element}) \quad (2)$$

This line intersects the T-axis at $T = T_{startout}$. Assuming $T_{element}$ is known (typically a small multiple of the cycle time), $T_{startstrip}$ can be determined from the slope of the line. This method of determining startup times is more accurate than examining the measured height of the steps because of the overshoots.

RESULTS OF VECTOR PERFORMANCE MEASUREMENTS AND THEIR INTERPRETATION

Execution times for typical vector operations were measured by timing 10^6 floating point operations for each vector length. A straightforward double loop was used, with the inner loop index running from one through the vector length N , and the outer loop index running from 1 through $10^6/N$. Because of the many repetitions of the outer loop, its startup time does not significantly distort the results. The time required for the timing function calls (CPUTIME under CTSS, SECOND under COS) was subtracted. Typical MFLOP rates for stride one measured for compiler versions 1.10, 1.11, 1.13, and 1.14 are presented in Table I for three vector lengths of $N = 10, 100, \infty$. Although CFT 1.14 shows a dramatic performance increase for short vectors with optimization options enabled (BTREGS, FULLIFCON, FULLDOREP) the rate for long vectors is lower than for CFT 1.10. Without optimization options, measured rates for CFT versions 1.11 through 1.14 are nearly the same and lower than those for 1.10 for all vector lengths. Figure 2 shows results for a simple vector multiply operation for CFT 1.14 with and without optimization options. The figure demonstrates that the optimizations reduce the startup time of the outer stripmine loop (zero intercept) considerably, while the startup time of each 64-strip (height of each step) remains unchanged.

Table II contains results for element times $T_{element}$, 64-strip startup times $T_{startstrip}$, and outer stripmine loop startup times $T_{startout}$ for the simple vector operations listed in column one. These data were obtained by measuring vector execution times for vector lengths equal to multiples of 64 (up to 512) and performing a least squares fit to the data points according to Eq. (2).

The results show that since the inception of the Cray X-MP two great changes in vector performance have occurred in CFT history: an increase in the startup time of 64 strips

by about 50% between CFT versions 1.10 and 1.11 and a decrease in the startup time of the outer stripmine loop by about a factor of four in introducing optimization options in CFT 1.14. Typical startup times for each 64-strip are 20 cycles with CFT 1.10 and 30 cycles thereafter. For the outer stripmine loop, startup times have been reduced from 110 cycles (with CFT 1.10) to about 25 cycles (with CFT 1.14 with options). It is noteworthy that the startup time of the 64 strips has to be added to that of the stripmine loop for short vectors at least once even if the vector length $N \leq 64$ and several times if $N > 64$. Thus a decrease in this time improves short as well as long vector performance. The question arises naturally whether we cannot have both short startup times for strips as well as stripmine loops.

We have examined code produced by the CFT compiler versions 1.10 and 1.14 for many loops. As an example, we present characteristics of the compiled code in Fig. 3 for a frequently encountered vector operation

$$V = S*V + S*V + S*V + S*V + \dots$$

where all V 's denote distinct vectors, all S 's distinct scalar operands. It can be seen that while CFT 1.10 uses a simple-minded approach to fetch operands as they are needed for the operations, CFT 1.14 prefetches as many vector operands as possible. This approach may work well for the Cray-1 (SAXPY now chains for the first time in Cray-1 history without hand coding), however, it is less efficient for the Cray X-MP with its multiple data paths. The philosophy results in additional non-overlapped startup times for fetches (plus associated non-overlapped A-register loads) and in some cases in delays due to lack of free V-registers. The X-MP has enough memory paths to supply its functional units with two operands on the fly; they do not need more. In addition fetches and operations chain. The most effective way of programming this machine is therefore the simple-minded approach used in CFT 1.10. The authors do not see any reason why this approach cannot be combined with the use of B and T registers to reduce startup times for the outer stripmine loops as demonstrated by CFT 1.14 so effectively.

CONCLUSIONS

Between CFT versions 1.10 and 1.11, a significant increase in 64-vector strip startup times has occurred. While CFT version 1.14 has reduced startup times for outer stripmine loops and thereby dramatically increased short vector performance, further improvements are achievable by eliminating the slowdowns introduced in CFT 1.11. This may necessitate different approaches to the scheduling of vector instructions for different computers in the Cray family.

CFT 1.10	CFT 1.14
Load	Load
Multiply	Load
Load	Multiply
Multiply	Load
Add	Load
Load	Multiply
Multiply	Add
Add	Load
.	Multiply
.	Add
.	.
.	.
.	.

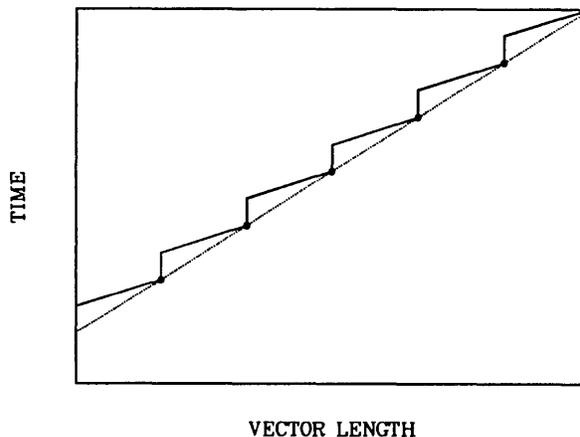


Figure 1. Plot of step function represented by Eq. (1).

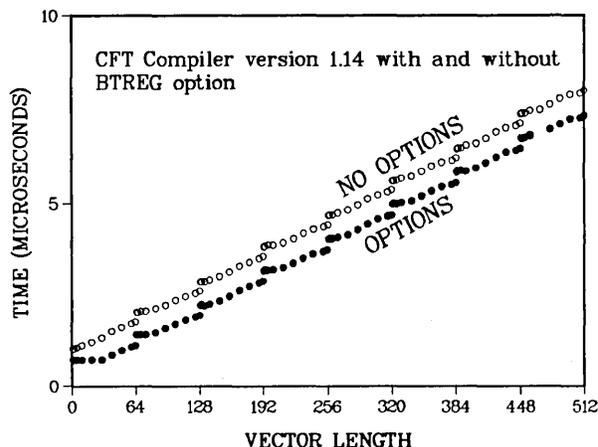


Figure 2. Measured execution times for simple multiplication in vector mode as a function of vector length.

Figure 3. Code produced by two CFT compiler versions for $V = S*V + S*V + S*V + \dots$

TABLE I
MEASURED RATES FOR SIMPLE VECTOR LOOPS
IN MFLOPS

Stride = 1	CFT 1.14 Options	CFT 1.14 No Options	CFT 1.13 Options	CFT 1.13 No Options	CFT 1.11 No Options	CFT 1.10 No Options
Vector Length = 10						
$v = v * v$	14	9	14	9	8	9
$v = v + s * v$	25	15	26	16	15	16
$v = s * v + s * v$	30	20	33	21	22	21
$v = v * v + v * v$	32	21	33	21	22	21
Vector Length = 100						
$v = v * v$	64	42	58	43	40	46
$v = v + s * v$	120	80	117	81	81	82
$v = s * v + s * v$	110	84	110	85	91	92
$v = v * v + v * v$	102	84	102	84	87	90
Vector Length = ∞						
$v = v * v$	72	72	72	72	67	80
$v = v + s * v$	145	145	145	145	144	160
$v = s * v + s * v$	124	124	124	124	140	140
$v = v * v + v * v$	116	116	115	115	131	135

TABLE II
START-UP AND RESULT ELEMENT TIMES
FOR SIMPLE VECTOR OPERATIONS

Stride = 1 Times in Nanoseconds	CFT 1.14 (Options)			CFT 1.14 (No Options)			CFT 1.13 (Options)		
	$T_{element}$	$T_{startstrip}$	$T_{startout}$	$T_{element}$	$T_{startstrip}$	$T_{startout}$	$T_{element}$	$T_{startstrip}$	$T_{startout}$
$v = v + s$	9.5	203	177	9.5	203	978	9.5	205	174
$v = v * s$	9.5	204	186	9.5	202	990	9.5	204	196
$v = v + v$	9.5	251	177	9.5	251	849	9.5	249	186
$v = v * v$	9.5	280	293	9.5	279	913	9.5	284	240
$v = v + s * v$	9.5	264	276	9.5	266	1082	9.5	275	263
$v = v * v + s$	9.5	288	269	9.5	286	1095	9.5	294	273
$v = v * v + v$	14.2	304	460	14.2	283	1121	14.2	281	456
$v = s * v + s * v$	19.0	331	199	19.0	332	1036	19.0	329	207
$v = v * v + v * v$	19.0	442	224	19.0	444	839	19.0	449	197
	CFT 1.13 (No Options)			CFT 1.11			CFT 1.10		
	$T_{element}$	$T_{startstrip}$	$T_{startout}$	$T_{element}$	$T_{startstrip}$	$T_{startout}$	$T_{element}$	$T_{startstrip}$	$T_{startout}$
$v = v + s$	9.5	205	931	9.5	206	863	9.5	171	984
$v = v * s$	9.5	205	949	9.5	206	875	9.5	171	955
$v = v + v$	9.5	247	798	9.5	345	925	9.5	190	974
$v = v * v$	9.5	282	856	9.5	341	924	9.5	189	1175
$v = v + s * v$	9.5	267	1038	9.5	289	989	9.5	190	1196
$v = v * v + s$	9.5	290	1027	9.5	340	995	9.5	219	1095
$v = v * v + v$	14.2	280	1065	14.2	284	1152	14.2	171	1434
$v = s * v + s * v$	19.0	330	1000	19.0	153	1096	19.0	152	1192
$v = v * v + v * v$	19.0	453	799	19.0	243	1090	19.0	207	1336

NAS KERNEL BENCHMARK RESULTS

David H. Bailey

Informatics General Corp. / NASA Ames Research Center
Moffett Field, California

Abstract

The NAS Kernel Benchmark Program, developed by the NAS (Numerical Aerodynamic Simulation) Projects Office to assist in supercomputer performance evaluation, has now been run on a number of currently available systems. This report briefly describes the benchmark program and lists the performance figures that are now available. Analysis and interpretation of the results are included.

Introduction

A benchmark test program has been developed for use by the NAS program at the NASA Ames Research Center to aid in the evaluation of supercomputer performance. This Fortran program consists of seven sections (referred to in this paper as *kernels*), each of which performs calculations typical of NASA Ames supercomputing. It is designed to provide a realistic projection of the supercomputer performance that can be expected on actual user codes.

The NAS Kernel Benchmark Program will not be described here in exhaustive detail. A more extensive description of the program, including a listing of the actual Fortran code, may be found in [1]. These seven test kernels were chosen from actual user codes currently being run on NASA Ames supercomputers and were included in the benchmark program with only minor changes from these user codes. All of the kernels emphasize the vector performance of a supercomputer - over 99% of the floating point calculations are contained in DO loops that are at least potentially vectorizable, provided the hardware of the computer includes the necessary vector instructions and provided the compiler being used is sufficiently sophisticated in its vectorization analysis. All floating point computations in the seven kernels must be performed with 64-bit precision (at least 47 mantissa bits).

Substantial care was exercised in the selection of these kernels to insure that none of them had any features that unduly favored any particular supercomputer design. The subroutines selected are all straightforward Fortran code, intelligently written for vector computation but otherwise neutral towards any particular model. An effort was made

to select a variety of calculations and memory operations. Some of the kernels contain vector memory accesses with only unit stride, while others have large strides. (The term *stride* refers to the memory increment between successive words stored or fetched from an array. For a real array indexed by the first dimension in a DO loop, the stride is one. For a real array indexed by the second dimension, the stride is equal to the first dimension.) Some contain loops with very long vectors (as high as 1000), while others contain loops with shorter vectors (the shortest in a time-intensive loop is 28). A brief description of each kernel is as follows:

1. MXM - Performs an "outer product" matrix multiply.
2. CFFT2D - Performs a two dimensional complex Fast Fourier Transform.
3. CHOLSKY - Performs a vector Cholesky decomposition.
4. BTRIX - Performs a vector block tridiagonal matrix solution.
5. GMTRY - Sets up arrays for a vortex method solution and performs Gaussian elimination on the resulting array.
6. EMIT - Creates new vortices according to certain boundary conditions.
7. VPENTA - Simultaneously inverts three matrix pentadiagonals in a manner conducive to vector processing.

Results

The NAS Kernel Benchmark Program has now been run on a number of large computer systems, and these figures may now be reported. Figure 1 lists the performance rates in MFLOPS (millions of floating point operations per second) for the various computers and compilers tested. The column headed NO. CPUs specifies the number of central processing units that were used in the computation (note that two of the Cray runs are for multiple CPUs). The column headed TUNING indicates the level of tuning performed (0, 20, 50, or unlim-

COMPUTER/ COMPILER	NO. CPUs	TUN- ING	KERNELS							COMP. RATE
			1	2	3	4	5	6	7	
Cray X-MP/12 CFT 1.13	1	0	131.0	30.2	36.0	71.4	5.2	74.5	21.5	24.5
Cray X-MP/12 CFT 1.13	1	20	131.0	82.8	51.6	71.6	102.0	107.4	112.5	88.2
Cray X-MP/12 CFT 1.14	1	0	130.7	30.7	35.3	71.6	50.1	82.0	21.6	43.9
Cray X-MP/12 CFT 1.14	1	20	130.8	82.0	50.4	71.5	110.3	97.7	116.4	87.7
Cray X-MP/22 CFT 1.14	1	0	136.5	45.7	47.0	73.8	65.1	81.4	37.1	59.9
Cray X-MP/22 CFT 1.14	1	20	133.7	89.3	60.5	77.2	118.2	97.6	115.5	94.4
Cray X-MP/48 CFT 1.14	1	0	136.0	45.9	59.8	82.3	95.5	84.1	30.5	61.9
Cray X-MP/48 CFT 1.14	1	20	136.0	85.2	66.7	79.6	115.5	103.0	124.1	96.4
Cray X-MP/22 CFT 1.14	2	20	272.0	175.3	112.0	141.2	219.4	193.2	238.6	182.1
Cray X-MP/48 CFT 1.14	4	20	536.8	330.9	205.0	273.3	395.3	396.6	483.9	349.1
CDC 205 F200PR1	1	0	128.0	12.7	5.5	10.8	3.2	5.9	10.8	8.9
CDC 205 VAST 1.21	1	0	116.6	12.5	24.2	8.0	21.3	61.1	9.4	16.1
CDC 205 VAST 1.21	1	20	129.8	49.5	108.4	14.5	72.1	76.9	52.8	44.7
CDC 205 VAST 1.21	1	50	127.8	57.4	108.3	135.7	75.0	76.2	67.4	82.9
Amdahl 1200 V10L10	1	0	465.1	11.1	42.2	88.5	38.3	214.5	7.3	22.4
Amdahl 1200 V10L10	1	20	497.2	106.0	95.6	88.0	127.5	214.9	202.3	139.1
Amdahl 1200 V10L10	1	50	500.9	106.5	96.1	91.3	127.4	220.5	202.4	140.8
Amdahl 1200 V10L10	1	unlm	499.2	162.1	96.7	124.5	150.9	219.4	232.2	174.7

Figure 1: NAS Kernel Benchmark Program Results (MFLOPS)

ited lines changed out of approximately 1000 total lines). The columns numbered 1 to 7 give the rates on the individual kernels, and the column headed COMP. RATE gives the composite MFLOPS rate. The composite rate is the total floating point operations performed in the seven kernels divided by the total of the seven CPU times.

Notes

Figures for more than 20 lines of tuning on the Cray X-MP computers are not shown because the rates listed for 20-line tuning are close to the maximum attainable level. The uniprocessor Cray X-MP/22 and X-MP/48 rates, especially the untuned figures, are slightly higher than the X-MP/12 rates because the X-MP/12 has slower memory chips and fewer memory banks than the multiprocessor models. The figures listed in the box for two CPUs on the Cray X-MP/22 and for four CPUs on the X-MP/48 are the result of running the NAS Kernel Benchmark Program simultaneously on each processor. These runs thus represent the total system throughput that can be expected in a multiprogramming environment. These runs do *not* represent multitasking, because multitasking means modifying a code so that separate parts of the computation run on separate processors, and the individual parts are coordinated. In these two cases the entire program was run on each processor without coordination, so they do not qualify as multitasking runs in any sense. However, they do provide a good estimate of the wallclock speedup that could be achieved if true multitask processing were invoked. Note that these figures are only 1.93 times and 3.62 times as high as the corresponding uniprocessor results. Memory bank contention prevents these rates from achieving a full two or four times speedup.

No tuned code figures are listed for the CDC 205 with the standard F200 compiler. This is because significant improvement in the performance figures would require utilizing the CDC Fortran explicit vector constructs, which are not allowed for this test because they are not standard ANSI Fortran. Using these explicit vector constructs and some other tuning, a composite performance rate of 84.1 MFLOPS was achieved, which is close to the 50-line tuning figure listed above for the CDC 205 with the VAST Fortran processor. According to CDC personnel, within a few months a new Fortran processor, based on vectorization techniques pioneered by Prof. Kuck of the University of Illinois, will be available for use on the CDC 205. This processor will likely yield higher performance figures than the VAST processor figures cited above.

The figures shown for unlimited tuning on the Amdahl 1200 Vector Processor system are actually based on approximately 400 lines changed. The Amdahl representative who performed this tuning is convinced that some further improvement in the composite rate is possible with additional revision of the code.

Tuning for the Cray runs was initially performed by the author. Subsequently a representative of Cray Research, Inc. reviewed this tuning and his suggestions were

incorporated for the final runs. Tuning for the CDC and Amdahl runs was performed by their own benchmark personnel with input from the author.

Analysis

The most striking aspect of the above table of figures is the great variance of the rates on the same kernel. Even on the same computer the rates vary dramatically. This spread indicates that even minor tuning can sharply improve execution speeds, and that an effective compiler is a critical part of system performance.

The most common tuning technique employed on the various systems was to change the dimensions of arrays in order to avoid the disadvantageous memory strides that result when arrays are indexed by other than the first dimension. For example, the dimension of one array in kernel seven was changed from (128,128) to (129,128) for both the Amdahl and the Cray tuned code runs. This change alone increased the performance rate of kernel number seven by a factor of 27 on the Amdahl 1200 and by a factor of 5 on the Cray X-MP/12. The second most commonly applied tuning technique was the insertion of compiler directives to force vectorization of DO loops. Most of the other tuning consisted of code revisions to avoid certain constructs that were not handled well by the system.

The process of tuning the code on the various machines disclosed several weaknesses of the various Fortran compilers. For example, one key DO loop in kernel five calls the complex exponential function. The Cray CFT 1.13 compiler vectorized this loop, but the vector version of the complex exponential function was merely a loop that iteratively called the scalar complex exponential function. As a result of this fact and the failure of the compiler to vectorize one other key DO loop, the untuned performance rate on this kernel was only 5.2 MFLOPS on the Cray X-MP/12. The difficulty with the complex exponential function was overcome in tuning by defining an inline version of the function at the beginning of the subroutine, as follows:

```
CEXP(Z) = EXP(REAL(Z)) *
$  CNPLX(COS(AIMAG(Z)), SIN(AIMAG(Z)))
```

This change, together with an altered dimension and a compiler directive, increased the performance rate on this kernel by a factor of 20. Both of the above mentioned shortcomings were rectified in the CFT 1.14 compiler.

Another feature of both CFT 1.13 and 1.14 discovered in the process of tuning is their failure to vectorize simple complex summation loops:

```
COMPLEX X(50), Z
Z = (0., 0.)
DO 100 I = 1, 50
  Z = Z + X(I)
100 CONTINUE
```

Such summation loops with real or integer variables are readily vectorized by both versions of CFT, but complex variable summations are not for some reason.

Coincidentally, the loop mentioned above with the complex exponential reference also proved troublesome to the Amdahl compiler, although for a different reason. The loop contained the line

$$ZZ = ZI - 1. / ZI$$

where each variable is of type complex. The compiler failed to vectorize this statement, and as a result the performance rate on kernel five was significantly reduced. For the minor tuning test this statement was rewritten using the complex constant (1., 0.). The statement was then correctly vectorized by the Amdahl compiler.

Except for the above mentioned details, both the Amdahl and the Cray compilers display a high level of sophistication. For instance, the CFT 1.14 compiler now includes a "conditional vectorization" feature. If the compiler cannot determine at compile time whether or not a recursive dependency exists in a DO loop, then the compiler generates both a scalar and a vector version of the loop, complete with an execution time test for safe vectorization. The Amdahl compiler appears to be even more sophisticated than the CFT compilers. It attempts to vectorize not only inner DO loops but also outer loops if conditions permit. In addition, if part of a DO loop resists vectorization, the Amdahl compiler vectorizes the rest, whereas the Cray CFT compilers generally vectorize a loop only if all statements within the loop are vectorizable. Another area where the Amdahl compiler seems to outperform the Cray compilers is in the vectorization of IF statements. CFT 1.13 vectorizes some IF statements if a certain option is specified, but only CFT 1.14 can vectorize IF ... THEN ... ELSE constructs. The Amdahl compiler vectorizes both of these constructs and even some IF ... GOTO statements, such as loops that search for the first occurrence of a given value in an array.

Conclusions

The three supercomputers tested have achieved high performance figures on the NAS Kernel Benchmark Program. The highest composite rate, 349.1 MFLOPS, was achieved by the Cray X-MP/48, which is to be expected since this was a four processor run. However, it should again be emphasized that this result is not true multitasking - the benchmark program was run simultaneously on each processor, and the results were added. Thus these results should be considered to be a measure of the overall system throughput capacity of the machine rather than the execution speed of a single job. A user could achieve comparable rates on a single job only by making the changes necessary to invoke true multitasking.

Comparing uniprocessor figures, the rates were closer, with all three systems achieving in the neighborhood of 100 MFLOPS on tuned code. Since each of the three systems has claim to the highest rate on at least one of the seven kernels, it is hard to make absolute statements about the relative performances of these systems. However, certain patterns can clearly be seen.

The Cray X-MP computers with the new CFT 1.14 compiler achieved impressive untuned performance figures, significantly higher than untuned figures on the other systems (considerably higher even than CFT 1.13). The Amdahl 1200, on the other hand, achieved very high rates on many of the kernels with some tuning, although the overall untuned performance was not spectacular. This is mostly due to the fact that the Amdahl machine is more sensitive to disadvantageous memory strides than is the Cray X-MP system. The CDC 205 is clearly capable of high performance rates (it had the highest rate of the three systems on kernel four), but it seems to require more tuning to achieve these rates. One reason that extra tuning is required is that the CDC Fortran compiler is apparently not as advanced as the compilers available on the other systems. This situation may be rectified with the introduction of more powerful Fortran processors on the CDC/ETA computers. Another factor in the CDC rates is the long startup times for vector operations. As a result, codes with vector lengths less than about 70 usually require revision (such as combining an inner loop and an outer loop into one loop) before the performance reaches its full potential.

The level of tuning that is the most appropriate for comparison depends on the nature of the supercomputing application. For a research and development application, the minor tuning figures may be the most important. Programmers in such environments usually apply some optimization techniques in their programs, but they seldom have time to perform massive tuning, especially on a code that is being continually revised. On the other hand, the major tuning or unlimited tuning figures might be more appropriate for a production computation application, where a single code might be used on a daily basis for years, and a large investment in optimization would be worthwhile over the long run.

Using the minor tuning figures (with the best available compiler) as a standard, it appears that the Amdahl 1200 Vector Processor has the highest uniprocessor performance rate, with about 1.5 times the Cray X-MP uniprocessor rates and about three times the CDC 205 rate. If major tuning is allowed, then the CDC figures are improved to nearly the Cray X-MP level, but the Amdahl figures are still about 1.5 times faster than the X-MP or the 205. If we consider the total system throughput with the minor tuning figures, then the Cray X-MP/48 is clearly the winner with about 2.5 times the throughput of the Amdahl. Similarly, the Cray X-MP/22 would likely achieve about 30 percent higher throughput than the Amdahl, although this comparison is closer if major tuning is permitted. These results must be considered tentative, since they could change overnight with the introduction of a more powerful version of the Fortran compiler on any of these systems.

Reference

1. Bailey, D. H., and Barton, J. T., "The NAS Kernel Benchmark Program", NASA Technical Memorandum, April 1985.

SSD USER EXPERIENCE SESSION

Mostyn Lewis

Chevron Oil Field Research Company
La Habra, CA

Approximately three years after the advent of the Solid State Device (SSD), it seemed appropriate to gather together some user experiences and some hopes. Conrad Kimball of BCS elaborated on experiences and Ronald Kerry of GM voiced hopes relevant to their impending delivery of an SSD. Mostyn Lewis of COFRC spoke of his site's locally developed SSD automatic preemption code and its latest enhancements and future.

COFRC SSD Code

Mostyn Lewis recapped the abilities of this locally developed major software effort to provide automatic SSD swapping. Over the last two years it was noticed that, in heavy demand, users were turning away from the SSD due to poor turnaround -- waiting for their chance at use (either for an initial allocation or waiting in "rolled out" state for their next go). Consequently, people shunned the SSD and went back to using disk, which although slower had a predictable turnaround. To help alleviate this situation, we changed the swap code to work on a "virtual SSD", i.e., disk. Hence, the user could execute all the time and in between SSD allocations execute in a "swapped out of SSD state". Optionally, at the user's choice, via JCL, the old regime of not doing anything in a swapped out state (i.e., being suspended) still exists -- this is for purists who wish to do timings and benchmarks.

COFRC will soon offer Cray access to its code so Cray can provide the same functionality in their equivalent (in COS 1.16?). Also, COFRC is willing to find suitable trades for other sites to use the code and General Motors are expected to use our code next year.

CRAY SSD SCHEDULING CONCERNS

Ronald Kerry

General Motors Research Laboratories
Warren, MI

The CRAY SSD provides definite application performance gains in an unconstrained environment (in terms of SSD demand). However, in an environment where there is significant competition for the use of the SSD, throughput can actually decrease.

I will discuss the specific concerns which General Motors Research Laboratories has regarding the use of the SSD in just such an environment. It should be noted that these concerns are purely speculative as we do not have an SSD at the present time.

General Motors Research currently has a CRAY-1S with 2 million words of memory and a 3-processor I/O subsystem. The applications which we run on our CRAY include engine modelling, aerodynamic, and structural analysis along with other automotive applications. These applications all require the computational power of the CRAY. They also require either very large amounts of memory or a large very fast I/O device.

Through benchmarking, we have shown that a large SSD can provide adequate application performance in a standalone benchmark environment. This led us to the decision to obtain a 128Mw SSD when we upgraded our CRAY to an XMP-24 in February 1986.

However, the expected performance gains could very well not be seen if excessive competition for the device results. The SSD is a very expensive device; it would also be wasteful if the SSD went virtually unused. We feel that significant enhancements need to be made to the CRAY Operating System software to take full advantage of the SSD in a multiprogrammed environment.

Some of the issues which we feel must be addressed by CRAY Research include:

1) control over how the SSD is used -- should it be used for memory roll images in addition to user data?

2) should a job be locked out until space is available on the SSD? -- should a job be rolled out of the SSD after a period of usage? -- it takes a very long time to roll out 128Mw worth of data!! -- what about a partial rollout of the data?

3) should data be split across the SSD and disk devices?

General Motors Research will probably obtain the SSD scheduling code written by Chevron in order to help alleviate some of the above concerns in the short term. In the long term, CRAY Research MUST step up to the responsibility of properly supporting a device which they sell! In the real world, many applications require large amounts of memory, or failing that, a large amount of very fast external storage. The SSD hardware has the capability to answer that need. Where is the software?

SSD EXPERIENCE AT BOEING COMPUTER SERVICES

Conrad Kimball

Boeing Computer Services
Bellevue, WA

OVERVIEW

Boeing Computer Services, a division of The Boeing Company, is currently running an X-MP/24 with an SSD-5 (134 million words) and 24 DD-29s. We are running COS 1.12, but with a 1.13 version of the IOS software. We have roughly 275,000 lines of local code distributed throughout COS, the IOS, the libraries, and the product sets. Part of this is our own SSD management system. We use the low-level Cray Research support of the SSD (device drivers, etc.), but we have replaced the higher level functions of allocation and scheduling. Our SSD management philosophy allows a single job to use up to 100 million words of SSD, with more available by special arrangement. To free up central memory, we have put CSP in the SSD.

SSD MANAGEMENT ISSUES

Goals

When Boeing Computer Services began planning for its SSD, we drew up a list of goals for the management of the SSD:

- o For sufficiently small amounts of SSD, there should be no need to declare any SSD resource requirements.
- o Reservation and allocation of SSD space should be deferred as long as possible, preferably until it is needed to do physical I/O.
- o When SSD space is reserved, minimize any excess reservation beyond the amount of SSD needed to do I/O.
- o Minimize the differences, visible to low levels of code or JCL, between SSD- and disk-resident datasets.
- o Avoid the need for users to know specific SSD device names.
- o Provide feedback to users about their usage of the SSD.

- o Interactive jobs must be able to use the SSD.
- o SSD accounting must be separate from disk accounting. In particular, an SSD residency integral must be computed.
- o Preserve any SSD datasets across a shutdown and restart.

Standard COS 1.12 SSD Facilities

Standard COS 1.12 supports the SSD as a generic resource, with several undesirable consequences.

SSD resources must be explicitly declared, regardless of how much or how little SSD space will be used. SSD resources can be requested only via the JOB statement. This means that:

Interactive jobs cannot use the SSD, since they have no JOB statement.

An unsophisticated user must be aware of whether any canned procedures use the SSD (and how much), and adjust the JOB statement accordingly.

SSD resources are allocated at the time a job starts. This leads to several inefficiencies:

A job is not started until there is enough free SSD space to satisfy its maximum SSD usage, regardless of how long the job may run before it actually uses the SSD.

Once a job starts, its SSD space is unavailable to other jobs, regardless of how long the job may run before it actually uses the SSD.

A job's SSD allocation monotonically decreases over time. This causes other inefficiencies:

Even though a job's maximum SSD requirement might occur late in the job, it must reserve (and leave idle) that much SSD space from the start of the job.

If a job uses SSD in two or more widely separated intervals, it must retain its SSD reservation until all SSD usage is complete. This may, in turn, require knowledge of how various canned procedures behave.

Low-level awareness of SSD usage is required. For example:

Individual ASSIGN statements must be used to assign the SSD to a dataset.

ASSIGN requires the user to know the site's SSD device name.

A RELEASE of an SSD dataset reduces the job's SSD reservation by the size of that dataset. Even though Cray Research is planning a no-reduce option for the RELEASE statement, the user must still be aware that SSD is being used, so that he can choose that option.

OPTION,STAT is the only tool that provides feedback about SSD usage, and it shows only cumulative statistics, when what is really needed is history of activity over time.

SSD usage accounting is not distinguished from disk usage accounting.

Boeing Computer Services' Implementation

In light of the goals that Boeing Computer Services set and the deficiencies in the standard COS 1.12 SSD support, we designed and built our own SSD management facility.

The maximum SSD that a normal job may use is 100 million words. With prior arrangements a job may use the entire 134 million words of the SSD.

No explicit SSD space declaration needs to be made for usage up to 10 million words. We chose the 10 million word cutoff for the following reasons:

Of all the jobs that run at Boeing, more than 90 percent use at most 10 million words of scratch disk space. Thus the vast majority of our customers' jobs can use the SSD without needing to declare it - all they must do is direct their datasets to the SSD.

Calculations showed that dynamic preemption of 10 million words of SSD would have acceptable performance (between 20 and 30 seconds, using non-striped DD-29s). As we do not yet have a dynamic preemption mechanism, the system provides an implicit declaration of 10 million words whenever a job tries to use the SSD.

When an explicit SSD declaration is needed, it can be done either by using the SSD parameter on the JOB statement or by setting the JCL variable 'SSD' to the number of blocks needed (e.g. SET,SSD=50000). This gives us several nice properties:

Compatibility with the standard SSD declaration on the JOB statement.

Canned procedures can make use of the SSD without the user needing to know about it or having to change his JOB statement.

Interactive jobs can use the SSD.

Reservation and allocation of SSD space is deferred to the last possible moment. Even though a job may have made an SSD declaration, the system does not actually reserve any SSD space until the job tries to perform a physical write on some dataset that is assigned to the SSD. The system (DQM) detects that no SSD has been reserved, and triggers the reservation mechanism. If not enough SSD is available, the job may be suspended at this point. When the last SSD dataset is released, the system releases the job's SSD reservation, until the next time that a physical write is done on an SSD dataset. As a result:

Jobs that use SSD will run, unhindered, until they actually need to use the SSD. At that point they may be suspended, but at least they have had an

opportunity to accomplish useful work in the meantime.

SSD space is not locked up and left idle between the time a job starts and the time that it needs the SSD.

A job's SSD declaration can be adjusted, either up or down, via the SET statement. One use of this allows a job to overestimate its SSD requirements, create all its SSD datasets, then reduce its SSD reservation to the amount actually in use. Of course, if a job increases its SSD declaration while it is holding an SSD reservation, there is a possibility of a deadly embrace with other jobs using the SSD. SSD preemption could handle the deadly embrace, but since we do not have preemption, we disallow any increase in the SSD declaration while a job has any SSD space in use (in effect the job must do the preemption itself).

To minimize the need for low level awareness of SSD usage, we implemented the following facilities:

Wherever a user can specify a device name, a user may specify a generic device name (as opposed to the name of a generic device). To assign a dataset to the SSD, a user need only use 'DV=SSD', rather than the full device name of 'DV=SSD-0-20'. In a similar fashion, to assign a dataset to any disk device, a user need only use 'DV=DD', rather than knowing about device names such as 'DD-A1-20', etc. To ensure datasets are assigned to distinct disk devices, a user can use 'DV=DD-ordinal', where 'ordinal' is an ordinal into whatever disk devices exist at that time (e.g., 'DV=DD-1', 'DV=DD-2').

For performance reasons, SSD datasets are assigned a default buffer size of 40 blocks.

A job may control SSD overflow behavior at the job level setting the SSDOVF JCL variable. If SSDOVF=TRUE, SSD datasets may spill to disk (unless a specific ASSIGN declares otherwise for that dataset). If SSDOVF=FALSE, the job aborts when an SSD dataset overflows

the SSD.

The OPTION statement was extended to allow a job to change its default buffer size, to change its SSD default buffer size, and to choose a default device for subsequent datasets. For example, OPTION,DV=SSD,SSDBS=20 would assign all subsequent datasets to the SSD, with a default buffer size of 20 (octal) blocks. While we were at it, we also propagated all relevant parameters (such as BFI and LM) from the ASSIGN statement to the OPTION statement. Of course, a specific ASSIGN statement can override any of the defaults selected by the OPTION statement.

To provide users with more feedback about their SSD usage, a local utility (DNLIST) can be used at any point in a job. DNLIST lists all the local datasets that exist, and if OPTION,STAT is turned on, also lists the OPTION,STAT information. Should an SSD overflow occur, the system informs the user (in the logfile) about which dataset overflowed, its size at the time of the overflow, and also the names and sizes of any other SSD datasets that may exist. At the end of the job, the system informs the user (in the logfile, again) of the high water marks of the job's SSD usage and scratch disk usage, and the times when they occurred.

To make things easier for our operations staff, we have changed shutdown to automatically flush the SSD if any jobs have SSD space in use. A subsequent restart will automatically restore the SSD, if necessary.

Finally, we modified COS slightly to allow CSP to be SSD resident. This frees up about 100K of central memory, without the performance penalty of putting CSP on disk.

PERFORMANCE OBSERVATIONS

In one performance study we observed that as buffer size increases, the sequential I/O transfer rate approaches an asymptotic value of 1 billion bytes per second. The transfer rate curve rises steeply at first and then levels off. The knee point is in

the vicinity of a 40 block buffer size, which attains approximately 75 percent of the asymptotic transfer rate (hence our default buffer size of 40 blocks for SSD datasets).

As we gained experience with the SSD, we noticed an interesting phenomenon. For many codes, the access counts of SSD datasets were almost exactly equal to the total blocks transferred divided by 1/2 of the buffer size - almost every physical I/O request was transferring 1/2 buffer. We believe that this is because the SSD is so fast that applications cannot keep up with it unless they are extremely I/O bound and use very big buffers. From this observation we have concluded that the 1/2 buffer threshold for physical I/O (embedded in the \$SYSLIB I/O routines) is counterproductive for the SSD. It seems that Cray Research has reached a similar conclusion, because an upcoming COS release will allow users to adjust the physical I/O thresholds of their datasets.

A flush of the 134 million word SSD, using non-striped DD-29s, with write-behind enabled, takes about 5 minutes. A restore takes about 10 minutes, since it writes the SSD, reads it back, and verifies that the SSD data is intact. Recently, however, another site discovered a bug in the write-behind logic that handles error recovery, so we have disabled write-behind. A flush now takes about 15 minutes!

HOW CRAY RESEARCH CAN HELP

Cray Research could do many things to help its SSD customers. Some of the issues that interest Boeing are:

- o Reduce the system overhead in processing SSD I/O requests. The new queued I/O scheme sounds like it will be efficient, but it will require application codes to change the way they do I/O. If queued I/O could be embedded in some library I/O routines, so it is transparent to the user, then it would really be useful. One of the nicest things about the implementation of SSD support is that the SSD can easily be substituted for disk, and standard I/O requests can be used.

- o Provide a high speed SSD preemption facility - both operator initiated preemption and dynamic (system initiated) preemption. Dynamic preemption should preempt only as much SSD space as is needed to satisfy the requirements of the higher priority job.
- o Provide more feedback to SSD users. As difficult as it may be to implement, what is really needed here is some way to plot, over the life of a job, how much SSD and scratch disk space the job uses, and the amount of I/O it does.
- o Fix the SSD scheduling algorithm to eliminate the 'dead' SSD space that occurs now.
- o Speed up the flush and restore of the SSD. One way might be to use striped disks; another might be to flush and restore only those parts of the SSD that are in use.
- o Allow sites to use the SSD as a high speed swap device. As main memories get bigger and bigger, so do the jobs that users run. As jobs get bigger, it takes longer to roll them out and back in. For jobs that occupy all of main memory, the system is essentially idle while the rollout and rollin occur. For example, a single job that uses all the memory on our X-MP, (about 3.9 million words), takes 10 - 13 seconds to roll out and another 10 - 13 seconds to roll in. Even with striped DD-49s it will still take about 3 - 4 seconds for a round trip. With the larger 8-million and 16-million word memories, the overhead of rollin and rollout could quickly get out of hand.

One way to alleviate this problem might be to use the SSD as a high speed swap device, or at least as a staging device. Using its high bandwidth, exchanging two 3.9 million word jobs in main memory would take less than a tenth of a second. The system could then migrate jobs between the SSD and disk, at its leisure. This could be made even more attractive by using the SSD back door to stage roll files between disk and SSD.

CONCLUSIONS

We have found that SSD performance is all that it is touted to be. Unfortunately, in an environment as diverse as that at Boeing Computer Services, the SSD is not as useful as we would like it to be. Much work remains to be done for Boeing to realize the SSD's full potential.

USER REQUIREMENTS COMMITTEE REPORT

Stephen Niver

Boeing Computer Services
Seattle, WA

The first part of this report deals with the results of the Winter '84 ballot. There were six items that were to be forwarded to Cray Research (CRI) for comment. The response from CRI is as follows:

COS Coding Standards

Following a well-established set of coding standards benefits both CRI and COS sites. Please publish these standards and modify COS in those routines that deviate from the standards.

Response: Cray does not plan to publish COS coding standards.

User Exits

User exits should be implemented at all important decision points in COS. These should provide hooks with a guaranteed interface at all places that users in general want to put modifications. Specification of this requirement in more detail would follow acceptance of the principle by Cray.

Response: Cray will consider customer requests for specific COS enhancements in support of User Exits.

Cray appreciates the need by some of its customers to implement local system code in COS to adapt the system to their specific needs. Since local code can make system support more difficult for both Cray and Cray's customers, Cray would give priority to User Exits which enhance the maintainability of the system and which benefit a large number of customers.

Installation Areas in COS Tables

Installations should be able to define and use areas within all COS tables. In some tables, it may be appropriate for CRI to set aside a guaranteed number of words. In other cases, the ability to increase the size of tables with an assembly-time table definition may be more sensible.

Response: Cray will consider customer requests for specific COS enhancements to provide Installation Areas in COS tables.

Cray appreciates the need by some of its customers to implement local system code in COS to adapt the system to their specific needs. Since local code can make system support more difficult for both Cray and Cray's customers, Cray would give priority to putting Installation Areas in COS Tables which enhance the maintainability of the system and which benefit a large number of customers.

System Tuning

As the COS system has become more complex, the ability to easily tune/modify the system assumes greater importance. It would be beneficial, therefore if CRI were to adopt a design direction that all tunable aspects of COS be parameterized and changeable via STARTUP directive or installation parameter as appropriate.

Response: Cray will consider customer requests for specific COS enhancements in support of System Tuning.

Most tunable aspects of COS are already parameterized. Cray would be interested in requests both for educational items related to tuning (such as the Job Scheduler Tuning Guide) and for tools in support of tuning.

Software Configuration

As the site configurations have become more varied, software has been written to support many diverse hardware and software features. Sites should have the ability to "configure out" that code that does not apply to their specific configuration.

Response: Cray will consider customer requests for specific COS enhancements in support of Software Configuration.

Cray has made enhancements for COS 1.15 in support of diverse hardware features. These include new CONFIG functions, such as the ability to up/down a CPU and a target CPU capability. Both STARTUP and CONFIG are being considered for further enhancement.

Support for "configuring out" code is not planned for COS. Although the Tape Queue Manager Task may be configured out of the system, other COS features are implemented within several tasks. Testing and performance considerations make configurability of these features at cross purposes with a reliable performance-oriented product.

Queue Management

Installations with large numbers of users can easily get into the position of needing to maintain large input queues. The development of network access to the Cray will lead to this becoming a universal experience. Running out of queue space leads to unrealistic operational problems. Expanding the number of possible queue entries to a related level would lead to an unacceptable large main memory commitment.

A queue management feature is required which would allow a much larger number of queue entries, including expansion to disc, dumping of queues and portions of queues. The feature must provide for the maintenance of relative priorities of items no matter where they reside.

Response: Cray has placed this feature in its planning cycle for future implementation.

SSD Management

Competition for scarce system resources dictates the need for an "intelligent" means for managing these resources. The system must make these resources easy to access (i.e., no hard specification on job card) yet manage the resource usage so maximum utilization does occur and resource deadlocks do not. COS already "manages" the CPU and main memory through the job schedule. Please extend COS to manage the SSD and buffer memory as well. Features should include (but not be limited to) the ability for system control (scheduling, allocation and deallocation/rolling) and operator-initiated control.

Response: A project for the SSD management capability described is underway and planned for release with COS 1.16. A design document has been completed for internal review.

Concurrent Maintenance

For sites committed to a 24-hour, 7-day-a-week production, system availability is a critical concern. To insure total system availability to the maximum number of users, some changes in the approach to overall system design is necessary. The design goal should be that when a system anomaly occurs, the failing component should be isolated so that a set of users are impacted rather than the whole customer set. In summary, the system design goal should provide that the whole production system should not be totally impacted when the unit failure occurs or while the failure

is being isolated, repaired, or when the failing element is returned to the production system. Some examples are the following:

- Disk drive/controller failures
- Single CPU failures within a multi-headed system
- Tape Drives

Response: The following areas have been identified for attention:

- Offloading disk data
- Add capability to allow diagnostics executed in privileged mode
- Create 'diagnostic' task
- Operator messages
- Access to channels
- 'Portioning' devices
- 'Portioning' memory
- Disk information requests
- CPU in maintenance mode

The first four items are complete and the remainder should become available progressively throughout 1986.

Summary of Ballot Responses

The appropriate Special Interest Committees will now work with CRI to assist CRI in specific implementation for those items CRI plans to consider.

The second part of the report covers the results of the recent (Summer, '85) ballot (Fig. 1). Following discussion in the User Requirements committee, it was recommended that CUG forward the top three items to CRI for comment. The lowest rated item, MODULA2, will be dropped; the remaining items will be carried to the next ballot. The two dotted lines on the chart graphically portray this. Those items above the top line are forwarded to CRI, those below the bottom are dropped, and those in between are carried to the next ballot.

SUMMER 1985 CUG USER REQUIREMENT SURVEY RESULTS

** RESPONSES SORTED BY TOTAL POINTS **

<u>NUM</u> <u>RESP</u>	<u>FEATURE</u> <u>TITLE</u>	<u>TOTAL</u> <u>POINTS</u>	<u>PERCENT</u> <u>POINTS</u>	<u>AVERAG</u> <u>RESP.</u>
28	PERMANENT DATASET PROTECTION	873.0	25.8	31.2
27	SCILIB EXTENSIONS	596.0	17.6	22.1
27	JOB DEPENDENCY	576.0	17.0	21.3

17	ENHANCED PDS MAINTENANCE	389.0	11.5	22.9
15	IMPROVED INTERACTIVE	275.0	8.1	18.3
19	JCL MULTI-TASKING	250.0	7.5	13.2
11	CRAY-TO-CRAY COMMUNICATION	231.0	6.8	21.0

2	CTSS SUPPORT	170.0	5.0	85.0
2	MODULA2	20.0	0.6	10.0
0	UNASSIGNED	0.0	0.0	0.0

Figure 1

SHORT PAPERS

SOME USER EXPERIENCE IN MIGRATING TO CFT1.14

Chris Lazou

University of London
England

The ULCC Environment.

The University of London Computer Centre provides large-scale computer services to members of the academic community throughout Britain. These services are based on a Cray-1S/1000 running COS1.12, CFT1.11 (old calling sequence) and an Amdahl 470V/8 running MVS SP1.3. Access to the centre is provided over X25 - based wide - area networks in conformity with the ISO model of Open Systems Interconnection (OSI).

The user community consists of postgraduates and university teachers, and totals over 6500 accounts. About 1500 of these accounts are Cray-1S users. Most of the users have to submit their work for a "peer review", to establish whether their work warrants a large-scale computer, before they are allowed to use the Cray-1S. This rather small system is overloaded and our users' requirements are one to two orders of magnitude larger than the computational capacity of the Cray-1S. The work simulated on the Cray-1S at ULCC spans the complete range of academic disciplines from physical to biological sciences on to humanities.

CFT Versions

Apart from CFT1.11 (old calling sequence) residing in the system, we also have CFT1.10, CFT1.11 (new calling sequence), CFT1.13 bug fix 2, CFT1.14 bug fix 2 on permanent data sets. Indeed many of the other bug fix versions are also there, which gives you an indication of the inherent instability of .CFT as a product. In addition to the Cray Products, we support the Cray Library, the mathematical libraries NAG and IMSL, and some 35 packages and other libraries including graphics.

The size of user programs run on the system, range from small development jobs, to large (several hundred thousand lines of Fortran statements) production jobs, partitioned to use as much of the Cray resources available.

Migration Path

As a matter of policy ULCC plans to effect upgrades during the summer when University teachers are free from undergraduate teaching duties. Since our user population is spread around the country, we have adopted the following migration path whenever we wish to upgrade a new version:

1. Document and distribute any external user changes, noting their possible impact on running programs to the user community.
2. Place CFT and associated products on permanent data sets and provide a procedure to access them.

3. Encourage application programmers at ULCC and the users at large to try new versions of CFT.
4. Generate new libraries on permanent data sets for users to access on a trial basis.
5. A stringent quality assurance exercise is initiated with the aim of assuring that all previous production programs still function correctly with the new versions (an impossible task with Cray software).

Problems encountered during migration.

Once the user community started using CFT1.14 the problems began to pop out of the woodwork. Our Cray analyst verified and submitted on ULCC's behalf, 7 critical, 5 major and 2 minor SPRs. There are 3 more known problems which are currently under investigation, not as yet isolated enough to establish whether we have to issue new SPRs for them.

The problems encountered were mainly due to the CFT compiler generating wrong code or the functions in the ARLIB library, have been "speeded up", by changing the algorithms, but with scant respect to accuracy. These problems were detected in large codes such as the LUSAS package (55K lines of code), a computational chemistry program (350K lines of code), crystallography package, econometrics, GAUSSIAN 82, and the NAG tests.

Remedies

With such spread of problems encountered at CFT1.14, ULCC was unable to upgrade last summer. Another problem which may be local to European sites, is that the response to critical SPRs by Cray Research is very slow. Even when code has been developed to solve the critical problem it is often not available to us for several weeks rather than days.

Recommendations

1. CRI should do more testing before releasing its products, if it wishes to preserve the confidence of the user community to their worthiness.
2. CRI should consider providing a mechanism for access of all current SPRs by all sites to enable installations to ascertain whether a problem they are hitting has previously been reported. This has the added advantage, for installation analysts, of providing material hints to assist them when trying to isolate problems in large systems. Some of these problems take days to isolate and any reduction of this unnecessary cost, would be appreciated.
3. CRI should consider publishing any changes to algorithms calculating floating point numbers results from mathematical functions and should try to conform to either IEEE or other suitable standards where available.
4. CRI must do better as far as CFT is concerned if it wishes to keep ahead of its competitors in this field.

SPECIAL INTEREST COMMITTEE REPORTS

NETWORKING AND FRONTENDS SESSION I

Dean W. Smith

ARCO Oil and Gas Company
Plano, Texas

The networking parallel session consisted of three talks by users on their experiences and desires regarding various Cray Research software products. Ronald Kerry's talk concerned General Motors' experiences installing and using the new Superlink/ISP product. Annabella Deck, from Chevron Oil Field Research, gave a presentation regarding Chevron's experiences running multiple frontend stations on a single CRAY and the problems they have encountered. I, of ARCO Oil and Gas, gave a talk on the networking of control information via the station messages facility.

CRAY INTEGRATED SUPPORT PROCESSOR
INSTALLATION EXPERIENCE

Ronald Kerry
Computer Science Department
General Motors Research Laboratories

The new Integrated Support Processor (ISP) is a software product that establishes a link between the CRAY operating system (COS) and IBM's MVS operating system. Through this link, CRAY users have local access to MVS data, device support, and data management services.

Having local access means that you can perform input and output on MVS datasets as if they were local to your CRAY job. You do not have to move the entire dataset to the CRAY first, a process that involves data staging and subsequent delays in program execution. Instead, a CRAY application program can go directly to the IBM device with no waiting.

I will discuss the installation experiences seen at General Motors Research Laboratories as part of the early support test program for the ISP product. This discussion will include:

- 1) the installation process;
- 2) the problems encountered during testing, along with the current status of the product;

- 3) some preliminary figures showing the performance of the ISP product, especially as it compares to the data staging techniques used by the current MVS station.

General Motors Research currently has a CRAY-1S with 2 million words of memory and a 3-processor I/O subsystem. The applications which we run on our CRAY include engine modelling, aerodynamics, and structural analysis along with other automotive applications. These applications all require the computational power of the CRAY. However, they are also all very dependent on information that is stored in our MVS database.

In September of 1984, Cray Research presented us with the idea of the ISP. We felt that it would benefit both Cray Research and General Motors Research to participate in a cooperative development and testing program. GMR expectations for the SIP were high and included:

- 1) ISP would relieve the pressure of storing large seldom used databases on local CRAY devices;
- 2) ISP would allow us to share data between CRAY and MVS applications;
- 3) ISP would significantly speed up data transfer between CRAY and MVS;
- 4) ISP would enable us to see much faster CRAY application performance.

The installation experience was divided into two distinct phases. The first phase I will call "advanced development." We ran into many problems. Most of these problems were found because of the fact that GMR computing environment is vastly different from the environment on which the ISP was initially developed. These problems included:

- 1) naming conflict with the interactive system productivity facility of TSO on MVS (SPF);
- 2) module reentrancy problems;
- 3) multiprocessor problems;
- 4) extended addressing problems.

The advanced development phase was carried out over three contiguous weekends after which we decided to wait until COS V114BF2 was stable enough to continue testing.

The second phase of the installation was the actual BETA test period. Several problems were found during this period, but none were as fundamental as the problems found during the advanced development phase. The BETA test period was completed in September of 1985.

Some of the major features and differences of the ISP include:

- 1) use of storage above the 16MB line for I/O buffers;
- 2) approximately 200Kb of SQA storage is used for control blocks;
- 3) a performance monitor is provided which runs as part of RMF;
- 4) a dump format routine is provided which runs as part of IPCS;
- 5) the default DF value is BB instead of the normal CB value;
- 6) output is binary zero filled instead of blank filled;
- 7) user job exits are provided which MUST be coded to enforce installation MVS JCL standards;
- 8) the documentation is in its infancy, but what is provided is very good.

Performance figures for the ISP are very preliminary due to the limited amount of time available in which to carry out experiments. We were able to sustain transfer rates of from 1 to 2 MB in each direction fairly easily while consuming 5% - 10% of a single 3084 type processor. If the TRACE option is turned on, the CPU utilization can go up to as high as 40%.

MVS block sizes and buffer sizes along with COS buffer sizes can affect the performance of the ISP greatly. In general, MVS block sizes had the biggest effect on performance.

In a worst case scenario, a COPYD of a 60 million byte dataset took three times as long using the ISP with DF=CB as the same operation using local CRAY datasets. However, if the time to fetch the input dataset and dispose the output dataset in the latter case is added in, the total time to do the COPYD using the ISP was three times shorter!

After all this discussion, it should be noted that the ISP is really a prototype product, with much improvement to follow. Some of the suggestions generated out of our early installation experiences include:

- 1) enable use of the hyperchannel for connectivity;
- 2) include the capability of writing SMF records in the performance tool;
- 3) general ISP recovery improvement (RAS);
- 4) installation options should be specified via a PARMLIB type arrangement instead of being assembled in;
- 5) allow MVS to initiate action with the ISP.

RUNNING MULTIPLE CRAY STATIONS AT
CHEVRON OIL FIELD RESEARCH CO.

Annabella Deck
Chevron Oil Field Research Company

At Chevron Oil Field Research Company (COFRC), users are free to use whichever computer they prefer. Choice is based on personal preference, the requirements of their application, the availability of disk storage, etc. For this reason, all general-purpose computers are connected to the CRAY, and there is a requirement that any CRAY job may access data on any frontend and submit jobs to run on any frontend regardless of the frontend of origin. There is a problem - because we have RACF on the IBM 3081 and 3033, and all datasets are protected. In addition, users like to be able to view and drop or kill jobs they have submitted to the CRAY from other frontends.

The problem is how to identify the user who submitted a CRAY job, independent of the frontend of origin, and in such a way that the user cannot change his identity.

When a CRAY was first installed at COFRC we had the CRAY MVS station and an

in-house VAX station and we used the TID field (of the Link Control Package) to identify the user. When the CRAY VAX station was installed, we found we could no longer use the TID field. We now use the userid of USR field. All stations have been modified to set the USR field in the dataset header for a job. This is set to the logon or userid of the user submitting the CRAY job. It is set independent of the user and cannot be changed. This field identifies the user, and is also used as the basis for all CRAY privileges and dataset ownership fields.

COS Changes:

ignore US field on JOB card;

US field on ACCOUNT card requires special privilege;

USR field propagation, DISPOSE - ok, FETCH/ACQUIRE - copy USR field to DSH, Station Messages - error in code setting USR in FSH.

COS Changes - SCP:

remove MF test for enter log file request;

when selecting a job for reply to commands STATUS, JOB, JSTAT, DROP and KILL, 1) remove MF test, 2) if requestor's TID = OPERATOR then allow request, 3) if requestor's TID = job's TID then allow request, 4) if requestor's TID = job's USR then allow request.

MVS Station Changes:

setup USR field in DSH for a job being submitted;

remove MF test for STATUS and JOB displays;

if no RACF slot then use USR field as RACF USERID;

if no TMS slot then build ACCTN field from USR.

VMS Station Changes:

set USR field in DSH for JOB and SUBDS to VAX USERID;

set US and UN fields in interactive logon segment to VAX USERID;

disable US parameter on interactive.

My request to CRI is that each station should implement a user exit whenever a

station slot record is read. The exit should be able to change the slot record or build its own if a slot record is not found. In addition, Cray should consider providing a standard ownership field independent of the frontend of origin of the job.

ENHANCED STATION MESSAGES SUPPORT

Dean Smith
ARCO Oil and GAS

The operation and system facilities of COS are often a less than a perfect match to the frontend system. The problem is not limited to different types of frontends. Because the systems that interface to a CRAY may themselves be dissimilar, this problem can be experienced with two nearly identical systems, and more than 1 frontend can guarantee an incompatible fit.

Reasons for the condition:

CRI is forced to develop solutions that have to be something to everyone.

CRI has had to develop system services (accounting, userid validation, password verification, dataset security services, etc.) that are alien to the host system, even though there exist counterparts on the host system.

CRI has limited resources with which to understand and address the problems of integrating CRAYs into our "alien" systems.

The result is often a "compromised solution".

My proposal is twofold: 1) enhance the station messages facility of SCP protocol to support many of the system facilities, 2) develop a flexible user exit facility on the frontends to act on the station message requests. In this way the various components and utilities of COS could utilize the station messages facility to obtain information, request validation, or communicate results back to the frontends. This facility should also provide for a user interface to the station messages facility.

The range of applications can span the entire system and user processing on the CRAY:

Accounting,
User Privileges,
PDN Access and Dataset privacy,
Tape Drive Allocation,
Job Scheduling,
Software/hardware Error Reporting,
CSP Exit,
ABEND Notification,
ABEND Recovery/Reprieve Processing,
Allocation of Local Datasets,
Job Initiation/Termination,
Operator messages

Judiciously implemented, the result could be a CRAY system which more closely resembles its host system to the user, the system operators, and system support personnel.

I believe there are advantages to both the user community and to CRI in this proposal. Some advantages to the users would be:

It would allow us to establish system-wide standards (one security system).

We could better utilize our own system personnel.

We wouldn't need two sets of processes to perform analogous functions on the different systems (one accounting system).

Additionally, there would be advantages to CRI:

Future enhancements and their implementation could be at the discretion of the site.

CRI could shorten the implementation time for new facilities by down-loading many of the responsibilities to the site support personnel.

CRI would not have to determine a "best" solution among various implementations.

LANGUAGES SESSION

Mary Zosel

Lawrence Livermore National Laboratory
Livermore, California

Four presentations were made in the languages session:

Peggy Boike, CRI - CFT 1.14 release
Wayne Anderson, LANL - Lisp
Kelly O'Hair, LLNL - LR Parser System
Karen Spackman, CRI - CFT77 (NFT) discussion

In the discussion of CFT 1.14, Peggy Boike, CRI, addressed some of the problems that were encountered with the CFT 1.14 release. Some of the beta testing procedures had been allowed to lapse. Peggy assured users that future compiler releases would go through extensive beta testing before release. Fixes for all major 1.14 problems reported before the time of the meeting, and most minor problems had been made and distributed to the sites.

Wayne Anderson, LANL, described the implementation of Portable Standard Lisp at LANL. This Lisp dialect was originally developed at the University of Utah. It currently runs under CTSS on LANL's Cray machines.

Kelly O'Hair, LLNL, presented a description of the LR system which is available for Cray machines. This parser system handles full LR grammars. It is written in standard Pascal and has been ported to multiple different machines and systems, large and small, including the IBM PC and SUN workstation. This parser system generates parser skeletons for the input grammar in the user's choice of several different languages: C, Pascal, Fortran77, CFT-Fortran, and LRLTRAN. One usually associates use of automatic parsers with compiler development, but at LLNL, O'Hair has found the main use is in developing interactive utilities. The system has been used to handle user interaction for the debugger, for the file transport utility, for a code analysis program, for a macro processor, etc.

Karen Spackman's presentation follows this summary.

CFT77 DISCUSSION

Karen Spackman

Cray Research, Inc.
Mendota Heights, Minnesota

First I would like to comment about the testing that we are doing on CFT77. We are very concerned about the reliability of our products and are doing what we can to ensure that the compiler is reliable before it is released. We do functional testing of specific features and run our own set of regression tests to make certain that we haven't introduced new problems. Currently our regression test base contains over one quarter of a million lines of code.

We will be taking CFT77 out to sites for beta testing before it is released. We are planning on two beta test sites for the CRAY X-MP release, one running COS and one running CTSS. We will also take the CRAY-2 version to a beta test site.

The fact that CFT77 is written in Pascal gives us an advantage in testing that our earlier products did not have. The testing department has written a coverage tool that works with products written in Pascal which measures how much of the code is exercised by the test set. We will be running this coverage tool against our existing test set to determine how good the coverage is now and what areas we need to concentrate on for future test development. We will also be running the coverage tool at the beta sites in order to find out how much beta testing improves the coverage.

We also do performance testing at Mendota Heights, and this is an area where we would like to do more. As well as running computational kernel codes, we have a set of "real" programs which we use for a performance measure. We would like to expand this set, and we need programs from you to do so. We are looking for actual user programs, not synthetic loops. They need to be well-behaved in the sense of being numerically stable and having answers that are fairly easy to check. They should be scaled down to run in one to five minutes of CPU time. Finally, because we are interested in tests to measure generated code performance, the programs should do minimal amounts of I/O, and the execution time should not be dominated by library routines. If you have programs that we can use, please contact me, Dick Hendrickson or

Jeff Drummond at Cray Research, Mendota Heights, Minnesota.

Question: Why didn't CRAY use Kuck's vectorizing preprocessor for the new compiler?

Answer: We are, of course, aware of the work of Kuck and his students since it is the foundation for much of the work that has been done in analyzing dependencies. We have certainly used many of the ideas from Kuck's work (and from Kennedy's at Rice University) in designing our approach to vectorization. We were interested in developing an integrated approach to the problem, however, that used the information gathered during the flow analysis done for scalar optimization and that took advantage of other optimizations done for scalar code. Consequently we want the vectorization analysis to be part of the compiler itself and not a separate preprocessor.

Question: Are there options to turn off optimization?

Answer: Yes. Full optimization is on by default. Control card options exist to turn off all optimization and to turn off just vectorization. When automatic partitioning for multitasking is available, this will also have a control card option.

Question: How does compilation speed compare with CFT?

Answer: We don't have a lot of information on this yet since the compilers that we build for testing in Mendota Heights have all of our debug code turned on, and this easily doubles the compilation time. We have done some preliminary timings with the debug code turned off and are seeing compilation times four to ten times slower than CFT. We are now analyzing where in the compiler we are spending the time and looking at what we can do in these areas. Right now global register assignment is taking a significant amount of time, and we are looking at changes that

should improve this substantially. Right now my best guess is that we will be looking at compile times more than four times those of CFT at the initial release.

Question: What about execution speeds?

Answer: Our preliminary scalar results have shown 10% to 30% improvements in runtime over CFT. Vectorization code is still being completed, so I don't have figures available for that yet. Our commitment all along on the project has been that CFT77 will generate code at least as good as that generated by CFT at the time of the release.

Question: What about the size of the compiler?

Answer: Again because of the large amount of debug code that we typically run with, I don't have a good feeling for what the size of the compiler will be at release. This is an area that we will be addressing in the next month, particularly in terms of segmenting the compiler. The data space used by the compiler does grow during compilation; there is no fixed limit on this.

Question: How does CRAY view continued CFT support in light of the CFT77 release?

Answer: Certainly we intend CFT77 to become the principal FORTRAN compiler for our machines; we will be retargeting CFT77 and porting it to all of our new machines. However, we want people to move to CFT77 because we have given you a better product with better performance. We certainly don't intend to force people to move from CFT to CFT77 by not supporting CFT. We will support CFT, for existing machines, as long as our customers find it necessary.

Question: Does the optimization we are doing for the initial release for the CRAY X-MP apply to the CRAY-2 also?

Answer: In general, yes. Most of the optimizations that are done are aimed at eliminating redundant operations which are redundant on any machine. Specific optimizations such as instruction scheduling have to be cognizant of the target machine characteristics, however.

Question: Will the user be able to compile for a different target machine than the one being compiled on?

Answer: Within the CRAY 1/X-MP line CFT77 will support a "CPU=" compiler option to allow retargeting. There are no plans at the present time to support cross-compilers between the CRAY 1/X-MP line and the CRAY-2.

Question: Will CFT77 gradually change into a FORTRAN 8X compiler?

Answer: My feeling right now is that we will support FORTRAN 8X with a separate compiler based on the optimization, vectorization, and code generation used in CFT77 rather than simply incorporating all of the FORTRAN 8X features into CFT77. FORTRAN 8X seems to be different enough that I believe users will want to have both compilers available concurrently for awhile.

Question: Will you be looking at loop unrolling for the CRAY-2?

Answer: I expect that we will be looking at loop unrolling. We won't have this available for the first version of CFT77 for the CRAY-2, but some of the early results from CAL code on the CRAY-2 indicate that we need to look at this for performance, probably for scalar loops as well as vector. Also some work one of our site analyst's did last year indicates that unrolling may pay off on the CRAY X-MP as well. We will certainly be investigating this in the next year.

Question: Can you tell us how you expect to approach automatic multitasking in CFT77?

Answer: Initially we will be looking at multitasking at the do-loop level, similar to microtasking except that we will have the compiler do the analysis to determine if the loop or program segment can be multitasked. In the next few years we will be looking at the whole problem of interprocedural analysis; this should let us expand the granularity of tasks that can be detected by the compiler. Our whole thrust will be to provide as much multitasking capability as we can without requiring the user to change his code. I expect we will introduce special syntax or directives only if we find significant ambiguities that we can't resolve.

SPECIAL INTEREST GROUP ON OPERATIONS

Gary Jensen, Chairman

National Center for Atmospheric Research
Boulder, CO

This meeting consisted of two sessions of the workshop. Attendance at the workshop was up by about 15% over the previous records. The facilities were outstanding and that made it quite enjoyable for all. We want to thank Gary Cross, the Operations Manager at Dorval, for arranging the great 'digs'.

I want to thank the speakers for the fine job they did in presenting their information to us in a most professional manner.

PRESENTATION DESCRIPTIONS

Andy Marien, Centre Informatique de Dorval, hosted a showing of a video tape presentation created by Cray Research, Inc., Central Region, titled "Installing an X-MP, from the view of Physical Plant Support". This video tape describes the problems that must be solved in order to have a smooth installation. The tape included many examples of how NOT to do it. The 'star' of the show is Andy and the Dorval Facility. We all want to thank CRI for the tape, and Andy for his comments and answers to the many questions.

Ray Benoit, Centre Informatique de Dorval, discussed "Networking at Dorval, Today and in the Future". Ray was the host of the entire CUG meeting and had been very busy throughout all of the meetings. Since he had to speak at most of the meetings, he had almost completely lost his voice. He gave an excellent presentation despite this problem. We want to give Ray a special thanks for the fine job he and his people did in organizing a smooth running CUG meeting. We will remember his raspy voice.

Gary Cross, Centre Informatique de Dorval, is someone none of us will ever forget. His presentation "Operation of the Dorval Computer Center" is included in these proceedings. Gary helped Ray Benoit set things up and was responsible for facilities, meals and parties. As you will read in his paper, he did get even with me at the party at Le Festin. Dressing up in the 1690 Governor's robes and playing that role was worse than any presentation I have ever had to make. I think my wife enjoyed it and thanks to her, I was not alone playing the Fool! Again, Gary, thanks for the fine job you did. I owe you one.

Dan Drobnis, San Diego Supercomputer Center, explained the plans, goals, and the current status of this new center. This center is funded by the Office of Advanced Computing, National Science Foundation. The center is now operating. Listening to this presentation convinced me that they will do well. Good luck, Dan. We hope you make yourself a regular participant at CUG.

Lou Saye, Cray Research, Inc., presented the Cray reliability statistics for the last six months. He did a fine job again, and we appreciate his participation. We want to also thank Gary Shorrel for his comments and help. We hope that they will continue to provide this information, in the future.

Fred Montoya, Los Alamos, described "The FOCUS System". His paper is included in these proceedings. We thank Fred for his continued support of the workshop. Fred has made several presentations in the past.

Thanks again, to all the participants and the Dorval staff.

COMPUTER OPERATIONS AT ENVIRONMENT CANADA

Gary Cross

Operations Manager

Environment Canada

Dorval, Quebec, Canada

Good afternoon and Welcome to CUG Montreal 1985. It's almost a relief to be here speaking to you today. I say that because as a member of the local arrangements committee, this talk will amount to my first break since Sunday morning.

When Ray Benoit asked me to participate on the CRAY local arrangements committee several months ago, I remember being ushered into an office and all the various categories which needed volunteers were written on a blackboard. Categories such as registration, finances, mailings etc., etc. When I was asked to participate in organizing part of CUG Montreal, I agreed for several reasons. To begin, I was fortunate enough to be asked first so I immediately chose by far and away the best category, food and entertainment. No way anything else came close. The second reason was that I could get away from the office and spend a few days meeting people and staying in a nice downtown hotel. So far so good. Thirdly I had the chance to spend large amounts of other people's money on food and drink and I loved it. Now everything was going ok, until I heard from Gary Jensen here. I was comfortably in the shadows spending other people's money and planning lunches and such, next thing I know Gary has convinced me to stand here for a talk on our Operations centre. Suffice to say I got more than I bargained for. Well I decided to get Gary back at his own game. The deal I made with him was that I would stand up here and speak for 25 minutes or so and Gary would consent to be the honorary governor at our supper tonight at Le Festin. I haven't told him yet exactly what that entails and I don't think I will. After all fair's fair.

As you can probably tell, I am not accustomed to public speaking especially in front of such a large and distinguished group, so please be patient. I'm going to do the best I can and I'd like to start by reviewing the outline of my talk. I'll be speaking in general terms about the makeup of our centre, explaining the hardware on site, personnel, shift schedules, plus some problems faced with managing this particular site. If there are any questions following my talk, please feel free to ask them. I only hope they're not in

the area of technical questions like bits, bytes or transfer rates because I tend to leave all that hard stuff to my support staff who, fortunately or unfortunately, are not present today. I don't know if I am departing from the norm in not really getting too heavily into hardware and software numbers and such, but I hope you will find it interesting nonetheless. So, with your permission I'd like to touch a few bases concentrating primarily on the makeup of our site from my point of view, that of the Operations Manager.

Let me begin by telling you just a little about who we are and what we do. Our shop is officially called the Dorval Computer Centre, or The Centre Informatique De Dorval, or CID. Most references to the Centre use the French acronym CID, as I shall. CID is a part of the Atmospheric Environment Service (AES), and as such is more or less the equivalent of the U.S. National Weather Center in Washington. In general, CID is responsible for producing weather related products for the country. This includes products for regional forecast centres in Canada, public radio and cable television stations, along with meteorological data relating to conditions for aircraft flights, farming conditions, and marine forecasts.

CID was the first CRAY supercomputer centre in Canada (installed in 1983), and it supports the AES. We are situated in Dorval, Quebec.

Between 1974 and 1982, a CDC Cyber 7600 was our large-scale computer, and this was replaced by a Cyber 176 as an interim measure until the installation and conversion to the CRAY 1S was complete.

It is worth mentioning here that in spite of the totally scientific nature of CID and its applications, we run a real time production shop. That is to say the meteorological products produced at CID must be distributed nationally under the constraints of very stringent deadlines. The operational weather runs executed on the CRAY must begin exactly on time and complete without incident, or nationwide delays are incurred and believe me we hear about it. So basically, CID is in business to produce

meteorological products which have a very high profile across Canada.

The CRAY 1S is currently front-ended by two Control Data Cyber 730 computers. These are used mostly for pre- and post-processing operations in scalar mode.

One Cyber 730 is used for real-time production processing and the second front-end for development or research work. If one of the 730's goes down, the remaining one switches to the production 730 to maintain our production deadlines. As a result, users on the development machine are out of luck until the second 730 is returned to service.

The development machine has eight Control Data 885 disk spindles, and the production machine has four Control Data 885 disk spindles. Each spindle has a capacity of 75 million words. The CRAY has a bank of twelve DD29 disk drives which provide a total of 900 million words.

There are two tape drives attached to each Cyber 730 and three STC tape drives attached to the CRAY 1S. All of these drives, while used for user testing, are in use most frequently for backing up permanent files and data sets. More on that aspect of CIDO in a few minutes. We also have, what we call, an input/output room adjacent to the main computer room where we process the paper output. The peripherals which cause most of the dust pollution in the computer room, were moved outside the main machine area to a spot where they wouldn't cause any dust/dirt problems. While it does make for a few extra steps several times per day for the operators, the overall benefit of having the machines which use ribbons and chemicals, away from disk and tape drives, is a definite improvement.

There are two CDC line printers, one attached to each of the 730's. On the average, we go through ten boxes of line printer paper daily which amounts roughly to 1.8 million lines printed per day, mostly test output for research. There are also two electrostatic plotters in the input/output room which produce graphics output, usually in the form of weather charts or related statistics. These, like the line printers, are constantly in operation as they strive to keep up with the mass of plotted and printed output queued on the Cybers waiting for their turn. Another room, also adjacent to the main computer area, houses all of CID's communications equipment such as our Datapac units, modems, tandem non-stop communications computers etc.,

etc. That's a very general overview of CID's hardware. Now I'd like to touch on the makeup of the different groups within CID responsible for supporting this equipment. CID is composed of four support groups each headed by a manager. Gerry Berlinguette is the chief of the centre which, of course, includes the four managers and their staff. The four groups are Communications and Graphics, Systems Support, User Services, and Computer Operations.

Communications and Graphics takes care of CID's networks and communications facilities. These tasks relate primarily to the Cybers and other communications equipment, as there are no interactive users hooked directly to the CRAY. Local and remote users (about 400) must first pass through the communications equipment to the front-end Cybers and then proceed to run jobs on the CRAY.

The second section is Systems Support responsible for installing and maintaining and troubleshooting all software on both the Cybers and the CRAY. You can take my word for it, that with a Systems Support staff totalling five persons including the manager, there isn't much spare time to be found in that group, or any other CID section for that matter.

CID also has a User Support Group responsible for processing and coordinating all users problems, requests, and sometimes demands. As you might already know, anyone who works in a user support capacity is long on patience, and if he or she lasts for a couple of years, usually qualifies for sainthood.

The fourth group is, of course, the Operations Section. I've saved the best for last, and I'll get into a few details about Operations in a second.

The number of persons in CID, responsible for all aspects of the Computer Centre and clerical administration total only 34. I know of some governmental centres half our size with twice the allocation of person-years. Believe me, when things aren't going well and we're pushed to the edge, I realize how much effort is required by these 34 people to settle things down and rectify any problems. It can get pretty hairy when several tasks or problems need simultaneous attention and there are only 34 people in the entire crew.

Furthermore, of these 34 person-years, a full one-third are in the Operations Section. That doesn't leave many people for software, communications, or user support. Well so much for self-

gratification.

Now a little bit about the Operations Group. There are currently 14 people in the Operations Section which is known as CIDO. The 14 are broken up in the following manner: one manager, two full-time day shift operations support staff, one tape librarian, and ten computer operators. The tape librarian is responsible for all the Centre's tapes, now totalling approximately 8,500 volumes. He handles all user requests directly, plus attending to all of CID's internal needs. He is a very busy person. He works five days per week, eight hours per day. The operators are not involved in handling user requests during his absence.

The two support staff members, who work directly for me, are responsible for the day-to-day needs of CIDO. Their primary duties include preparation of operator shift schedules, scheduling all work on the three mainframes, attending daily manufacture meetings, controlling all user disk space allocations, preparing operator instructions and procedures, as well as preparing CID stock and supplies contracts and coordinating delivery, storage, and allocations of this stock.

They both work five days per week, eight hours per day, and are available 24 hours per day, seven days per week via electronic pagers for calls directed to CIDO from the operating staff, manufacturers, or other CID sections. As a matter of fact, the Systems Section and the Communications Section also carry pagers for the same purpose. The User Services Section has an automatic answering system to record user inquiries after normal business hours. So in effect, personnel from each of the four CID sections are on 24-hour standby.

The ten computer operators (one is temporary) are obliged to work many shifts since CID runs a 24-hour day, 7 days per week, 365 days per year operation. We have been using a 12-hour shift cycle since 1974. I'll give you a quick idea of how it works. There are two operators per 12-hour shift. One is the shift coordinator, and the second is the computer operator. The shift coordinator is responsible for the shift and consequently is one level higher than the computer operator. That is basically the only major difference between their functions. This is because the work load requires that they function as an absolute team, meaning one must be able to handle the duties of the other and vice-versa. So, through evolution, they both perform the same duties on shift. That was not the way the

original job descriptions were designed for the staff some ten years ago, but as the centre got bigger and the responsibilities grew, the operating staff remained static at two per shift. Therefore, the duties for each, which were once well separated and defined, are now more or less melded together.

Besides monitoring all systems and performing the usual tasks associated with operating in a multi-mainframe environment, each shift is required to log all hardware, software or environmental interruptions as well as any other incidents that may occur. All events are logged on specially designed forms which become the main input for meetings held daily with a representative from each CID section as well as from both computer suppliers. Each incident is discussed in a round-table format, and is assigned to one of the representatives for action. Follow-ups are also done and various reports are generated from these meetings as all pertinent data is entered into a data base on the Cyber front-ends.

We do not have the luxury of assigning specific tasks to the shift coordinator or to the operator. Given all the equipment which requires monitoring and the paper which has to be cut and the tapes which have to be mounted, there is simply no way we can now split up their duties unless more staff is hired. But each shift has an operator and coordinator and, as I said, there are two people per 12-hour shift.

As I said previously, the 12-hour shift schedule has been in effect for over ten years and frankly, from the operators' point of view, is the best thing that ever happened to them. I was an operator in CID for over ten years and I've seen many schedules come and go, and having worked them all, this one is tough to beat. The SKED works this way, and is identical for the ten operators. Government workers, regardless of their shifts, operate on a 37.5-hour work week.

For the operators each of their shifts are 12.25 hours long, either from 7:45 A.M. to 8 P.M. or 7:45 P.M. to 8 A.M. The fifteen extra minutes is for a debriefing period between shifts. Each operator works four of these 12-hour shifts, starting with two night shifts, then a 24-hour break after completing the second of the two night shifts. He/she then works two 12-hour day shifts and is off for five days. So that's the way it works, four on, five off, four on, five off, and so on.

The SKED is based on a 56-day rotation and at the end of the 56 days each operator "owes" the schedule 8 hours. This is usually made up as a project-day during one of the days off. Besides affording the operators with copious amount of time off, they are also available to work plenty of overtime, that magic word. Due to tight government restrictions on the hiring of people at our site and in general, throughout the government, overtime is, at CIDO, a very real requirement. Each time our operators take annual leave or "book off" sick, that shift must be filled with overtime. This applies to one day off, or one month off and we've experienced both many times. Seeing as how all of our operating staff have four week holidays, overtime payments cost CIDO a small fortune. What helps to run up our O/T bill is that when an operator is on his or her 5 days off and comes in for a 12-hour O/T shift, the second to 5th days off are paid at a rate of double time, which equals 24 hours at their regular hourly rate.

During the peak summer months of July and August, we average 25 to 35 overtime assignments per month, sometimes more, seldom less. We are very fortunate, in a way, that all of the operators are ready, willing, and able to work large amounts of overtime because, even if only one or two balk from time-to-time the on-site staff must work a double or 24-hour shift. So, you can see that given our current hiring constraints, if one or two operators refused all O/T offers, we would be in quite a bind.

CIDO just doesn't have the required cushion of person-years to help reduce our O/T budget. The operating staff has always been receptive to the requests for O/T work and continue to be. This does help considerably when producing shift schedules, especially during summer and holiday periods.

One final point about the CIDO shift schedule. Every three months the operators' cycle is rotated from the shift supervisors' cycle. It just doesn't work to leave two people together for more than three months. For each team that loves working with each other, there are at least two other teams who really don't get along all that well. The only alternative is to keep them all moving along with a three-month rotation. There are a few long-term problems associated with the operating staff which are no fault of theirs, but more related to the acquisition, or better yet, non-acquisition of staff. Of the nine full-time operators on staff in CIDO, the most

junior person has about ten years' service in our shop, not just government service but ten years operating our computers. The negligible staff turnover is due, in part, to the lack of any career paths for the operators. There is just nowhere for them to go within CID. Person-years are just not available, hence training programs in other sections for temporary periods do not exist. Further, again due to person-year shortages, I cannot spare even one operator for training stints either within or outside of CIDO. Therefore, it is very difficult to motivate the operating staff to do anything but operate. I must add here that a large majority of the operators are self-motivated. They genuinely take pride in their work and do a fine job.

One big plus about having a veteran operating staff is that they are as up-to-date and aware of our methods, practices, and procedures as anyone. I can also rely on them to learn new instructions rapidly, and they often point out ways to improve on existing standards.

One compensating factor, though, is the salaries paid to the operators. We are a union shop, and as such, the salaries, even by american-dollar standards, are hard to beat. The base salary structure, coupled with the number of overtime hours worked make for a generous yearly salary. It's not the greatest motivating factor in the world, but it does keep the complaints down.

The operators belong to one government union and the day shift workers belong to a second union. From a management point of view, there is certainly nothing to fear from either union. The common bond between the two unions is to see how much money in union dues they can remove from our paychecks and how fast they can do it. They are not at all what you would call militant, so no difficulties are caused by the presence of unions on site.

You might now have the idea that staff shortages are a major problem at CID. Well, yes and no. I think we could use a few extra people here and there to help us push forward and expand more rapidly, but I am not implying that we are lagging behind in our work or are unable to properly function. CID has adapted extremely well over the years to a pared-down staff, and it is certainly to our credit that we have progressed as far as we have in a relatively short period of time.

I'd just like to now touch on some miscellaneous topics to highlight a couple of Operations' tasks. They might provide some useful

comparisons to your sites.

CIDO is responsible for the archiving of all permanent files on the Cyber disks and all the data sets on the CRAY disks. We have incremental dump routines on both the CRAY and Cybers which are executed daily. Full dumps of all files on the CRAY and the Cybers are done once per week. Disaster dumps of all disks are done once per month and stored off site for a period of a year. All the files, whether incremental, weekly, or monthly are dumped to magnetic tape which, needless to say, requires constant recycling and manipulation by our support staff and tape librarian. CID's Systems Group has recently provided Operations with a CRAY incremental dump package. Prior to that, we were dumping the complete disk catalog once per day which ate up 1.5 to 2.5 hours of time. Progress is being made.

Preventive maintenance on the CRAY is performed twice each week (Mondays and Fridays) with two hours allocated per period. This was reduced from five times per week during CRAY-acceptance to three times per week, and then to the current schedule of twice per week. Preventive maintenance on the Cybers is twice per month for each front-end. Each period lasts two hours. One week the development machine is under P.M., and the following week the production machine is under P.M. When production is on P.M., the production disks and software are switched to the development machine so the production system is never down for P.M. periods. CID does not permit changes of any kind (hardware, software, temporary, or permanent) to be performed on any of our computers by engineers, analysts, Operations, or System personnel until proper documentation is supplied to CIDO and approved by the appropriate manager or managers. Once approval is given, Operations then schedules the time on the designated system, and the users are then informed using computerized bulletins. There are also minimum times required before anything is scheduled, depending on the impact of the change.

The immediate future for CIDO looks quite interesting. The CRAY is scheduled to be replaced by the end of 1986. The two Cyber front-end computers will be replaced in the early part of 1986. In spite of the fact that nothing seems to be permanent here but change, it certainly makes for interesting times and produces ever different problems to solve and situations to handle. That, to me, is what managing an

operations shop is all about. It keeps changing and evolving almost right before your eyes.

Lastly, due to the sensitive nature of CRAY supercomputer technology, full-blown security equipment and procedures have been set up at CID. The features include a 24-hour security guard team, ID cards for all personnel as well as visitors and service personnel, selected entries to controlled areas using electronic card access, security cameras and video recorders, and Halon fire retardant systems. The features are constantly under scrutiny and enhancements are often made. It did take time for some people to adapt to the move from no security although the staff adapted well.

Well, I think I've gone on long enough. I must admit that I rather enjoyed the experience. What I really hope is that I was able to shed a little light on the operations in's and out's at the Dorval Computer Centre.

FOCUS AT THE LOS ALAMOS NATIONAL LABORATORY

Fred J. Montoya

Los Alamos National Laboratory
Los Alamos, NM

ABSTRACT

During the past three years, the Computer Operations Group at Los Alamos National Laboratory has operated the *FOCUS* System (Facility for Operations Control and Utilization Statistics). *FOCUS* is responsible for production control, load leveling, and status reporting. This paper describes the operation of *FOCUS*.

INTRODUCTION

The Computer Operations Group (C-1) at Los Alamos National Laboratory operates the Central Computing Facility (CCF). The Group consists of 68 people including the Group Leader, the Associate Group Leader for Operations, and a Supervisor responsible for Special Services. The operators are divided into three teams, A, B, and C. Each team has a Supervisor, a Deputy Supervisor, three Lead Operators, and twelve operators.

OUR ENVIRONMENT

The CCF houses the following major computers: two CRAY-1As, two CRAY-1Ss, one CRAY X-MP/2400, one CRAY X-MP/4800, three CDC 7600s, three Cyber 825s, one Cyber 855, and one Cyber 176. One IBM 3083 and one IBM 4341 control the Common File System (CFS). The CFS is used as a data storage device by all of the worker computers. A large array of mini-computers are used as gateways to provide Integrated Computer Network (ICN) service from remote computers through the XNET System, or as hosts to external networks such as ARPANET and TELENET.

The network is divided into three partitions: Secure, Administrative, and Open. This partitioning avoids having duplicate systems for each level of computing, but it adds to the complexity of the operation.

The CCF is operated 24 hours a day, 7 days a week, 365 days a year. We schedule a 48-hour, holiday shutdown at Christmas/New Years, and a two-day maintenance shutdown twice a year (usually during a three-day holiday weekend) in the fall and in the spring.

FOCUS OPERATIONS ENVIRONMENT

FOCUS is a component of the ICN that automates production control, station reporting, and performance measurement. *FOCUS* currently operates with a primary and secondary controller (a VAX 780), using periodic software backups to reduce the effects of failure. Reliability and availability are very good, but our goal is continuous, error-free operation.

The *FOCUS* System's primary function is to schedule production work on all CRAYs and 7600s. Production is defined as the mode of jobs that are scheduled and run by the computer center on behalf of a user. A production mode job is run independent of the presence of the user.

The method of scheduling is based on several objectives of the system.

1. The primary objective distributes the CRAY production resources on a continuous basis to the major divisions of the Laboratory according to the Director's allocations.
2. A secondary objective allows organizations control over which jobs are run within an organization's allocation.
3. The third objective allows organizations flexible control for "saving" and "overspending"

allocations to handle workload fluctuations.

4. A fourth and final objective optimizes the use of the CRAY production resource.

An organization is allocated the CRAY production resource in proportion to its requirements. The allocation is transformed into a fraction of the resource, not as a fixed amount of service. This assures that variations in the available resource are distributed in an equitable manner. Because fluctuations in workload make it impractical to keep all organizations exactly serviced according to allocations, a history of usage is kept to force long-term usage to correspond to allocations while allowing short-term fluctuations.

An organization needs some flexibility and is able to control and manage the workload within the organization. This allows them the ability to sub-allocate, the ability to order jobs or define ordering criteria, the ability to time job-leveling factors within the organization, and the ability to control the "saving" or "overspending" of the allocation.

From the operations side, an effort is made to improve machine performance. This implies that the scheduler will monitor the utilization measures on the production machines and schedule jobs to a machine when it appears that utilization can be improved. A machine is not permitted to go idle when there is eligible work. Furthermore, maintenance schedules, special conditions, and end-of-shift conditions can be anticipated and accommodated efficiently.

FOCUS has three production shifts (DAY, NIGHT, and WEEKEND/HOLIDAY) that are allocated, charged, and historically recorded independently of each other. The scheduling implementation requires the same master queue structure for all three shifts; however, each shift is separately allocated.

The batch subsystem on the CRAY computers operates as slave to the *FOCUS* scheduling. Although *FOCUS* controls the initiation of each job, the running of jobs is controlled by the batch subsystem. The intent is to have the batch subsystem run the jobs with higher CTSS priority given to the ones initiated first. However, multiple jobs may be running on a given machine at any one time and the dynamic nature of the job will produce a multi-programming mix of production jobs on the CRAYs. *FOCUS* determines dynamically the degree of multi-programming for each CRAY based on production parameters and machine utilization statistics.

Based on the above, jobs are scheduled to a machine when it needs work. When the queues are searched for the next eligible job, the characteristics of the machine being scheduled must be considered. Some jobs are ineligible because they require more than the maximum memory of the machine, their time limit would extend the current committed time on the machine beyond the current period, or the job specified a specific machine. In addition, a job may be ineligible because it is dependent on another job that has not been completed successfully.

FOCUS MENUS FOR THE OPERATOR

FOCUS is an automatic system; however, the system has to be monitored by an operator. The operator has a menu that offers many tools to effectively and efficiently operate *FOCUS*. Most commands can be entered with the touch of a finger on the touch screen of a VT100 terminal.

The following are the menu options that are available to the operator.

- FOCUS - Menu of anything that has to do with *FOCUS*.
- INFORM - Status of all worker computers, also allows the operator to select an individual computer.
- JOB COMMAND - Menu for scheduling jobs.
- JOB STATUS - Status of all jobs in all the worker computers.
- MACHINE COMMAND - Menu allowing an operator to change parameters, time limits, memory limits, and set dry up.
- MACHINE STATUS - Overall look at the status of jobs that are running or waiting to run.
- PRINT CARRYOVER LOG - Listing of the carryover from the previous production period.
- PRINT JOB LOG - Summary and status of all jobs submitted during the previous production period.
- PRINT SUMMARY - Summary for any month of production on an individual machine, or a complete summary of all machines.
- QUEUE DISPLAY - Master queues of all user divisions.
- SYS ACTIVITY - Lets the operator display which process in the system is using the CPU.

- UTILIZATION - Current shift production report for all CRAYs and 7600s.

- GATHER - Responsible for updating and reporting the number of users and CPU utilization.

- DEBUG - Used by system personnel.

- DISPLAY - Runs *FOCUS* gather and displays information in the color monitor.

- CURRENT PROCESSES - The given status of all processes.

- PHONE - Displays the office phone number of all ICN validated users.

- CURRENT TIME - 2400-hour military clock. WWVB, National Bureau of Standards Radio Station, Denver, Colorado.

- ACKNOWLEDGE - Still in the development stage. Will alert the operator that a message exists on a worker (for example, "waiting on tape mount").

- HELP - Help package for operator.

- QUIT - Exits operator from *FOCUS* and the terminal becomes a regular user terminal.

Another tool in the *FOCUS* System is the automated trouble log. Instead of writing trouble logs, the operator enters all information into a VT100 terminal. Every weekday morning, with a simple command, management can receive a print-out of all worker computer and equipment malfunctions that have occurred during the past 24 hours.

OTHER USER OPTIONS

The user has other options with *FOCUS* that help operations. A user can sign on to a user terminal and access *FOCUS*. The user can view the job queues and get a good idea as to the status of his jobs. Another feature is the touch tone telephone call. The user calls a certain telephone number and a digitized voice will answer giving the user instructions to touch tone in his user number. *FOCUS* will then scan the queues for the user's number and respond with the status of his jobs.

CONCLUSION

FOCUS made Computer Operations more efficient. We are now able to operate the facility with fewer people, yet our throughput continues to increase. Before *FOCUS*, we were using five operators on each shift for a total of fifteen operators to operate five CRAYs. At present, we are using two operators per shift on *FOCUS*. Even if we add more worker computers to the ICN, the *FOCUS* staff will not increase.

The user organizations are responsible for the allocations and the scheduling is accomplished by using a centralized control machine. The primary advantage of centralized scheduling and control is that several worker computers can be scheduled, allocated, and viewed as a single production resource.

MULTITASKING PERFORMANCE WORKSHOP SUMMARY

Ann Cowley

National Center for Atmospheric Research
Boulder, CO

Three papers were presented in the workshop. The abstracts are included here, and the papers by David and Dent are included in their entirety. Koskela's paper was not submitted for publication here.

MULTITASKING THE WEATHER David Dent - ECMWF

The ECMWF Model uses both cpus of a CRAY X-MP/22. Performance figures will be presented together with measurements of overheads and inefficiencies. The repercussions of moving to a CRAY X-MP/48 will also be discussed.

VECTOR USE AND CONTENTION MEASUREMENTS Rebecca Koskela - LANL

Performance measurements for parallel and vector processing are reported for the CRAY X-MP supercomputers at Los Alamos National Laboratory. The measurements are made with the CRAY hardware performance monitor. Three kinds of measurements are made: (1) we measure the percentage of vector instructions executed system-wide, (2) for parallel processing, we measure the amount of memory contention in the CRAY X-MP shared memory architecture for 2, 3, and 4 processors, (3) we also measure the percentage of time a processor is blocked waiting to execute in the shared operating system because another processor is executing in it.

CMTS - A CRAY MULTITASKING SIMULATOR Jacques David - CEA-Limeil

CMTS is a CRAY Multitasking Simulator which can run on CRAY-1 or X with CFT 1.10/COS 1.11 and later releases (ALLOC=STATIC), or on CYBER (NOS/BE - NOS-SCOPE) systems. It can be used for testing and debugging multitasked applications and gathering various statistics (Locks/Events/Speed-up...).

MULTITASKING THE WEATHER

David Dent

European Centre for Medium-Range Weather Forecasts
London, England

INTRODUCTION

The European Centre for Medium Range Weather Forecasts has the dual responsibility of:

- a. Carrying out research into numerical weather prediction, and
- b. producing a 10-day forecast on a daily basis to a strict operational timetable.

This second activity has generated the need for the weather model to execute as efficiently as possible on the available hardware. This report outlines the methods which have been employed to allow the model to utilize multiple central processors of a CRAY-XMP and presents detailed timings which indicate where inefficiencies exist.

HISTORY

The present production model has been developed over a number of years and is used both for research and operational forecasting. The model uses spectral techniques and covers the complete globe. It consists of about 100,000 lines of Fortran and requires work files to hold its data. The code is independent of the spectral truncation chosen, i.e. the data resolution.

The model first went into daily production in 1983 at resolution T63, executing a 10-day forecast on a CRAY-1A in 5 hours. The same resolution model was moved to a CRAY-X22 in 1984 and executed on one CP in 3 hours, using the solid state storage device (SSD) for the work files. In 1985, the resolution was increased to T106 and currently executes in 5 hours, 15 minutes using both processors of an XMP-22.

ECMWF CRAY-XMP CONFIGURATION

From the point of view of the spectral model, the principal characteristics of the CRAY-X2200 installed at ECMWF are:

- 2 Central Processors
- 2 Mwords of central memory
- 16 banks of memory
- 16 Mwords of SSD

80 Mwords/sec memory to SSD transfer rate

COMPUTER RESOURCES USED BY THE SPECTRAL MODEL

At resolution T106, the single-tasking model requires:

- 1.5 Mwords of central memory
- 15.3 Mwords of SSD

There are 3 work files, totaling 15.3 MW and transferring 30 MW of data to/from SSD per time step.

Putting files on a device with such a high transfer rate to/from central memory allows I/O to be carried out synchronously without much overhead. This reduces the central memory requirements for buffer space and costs less than 3% of the elapsed time for a 10-day forecast.

MULTITASKING INTERFACE

The following facilities available in the Cray multi-tasking library are used in the model:

- CALL TSKSTART
- CALL TSKWAIT
- CALL LOCKON
- CALL LOCKOFF

These tools enable tasks to be started and synchronized, and critical areas of code to be protected against simultaneous execution.

GENERAL STRUCTURE

The model is organized into 2 scans over the data, as shown in Figure 1. Within each scan, there is a loop over all latitude rows (160 for the T106 resolution). Between scans is a smaller area of computation associated with diffusion and semi-implicit calculations. The loop over time steps is repeated 960 times for a 10-day forecast. However, every 12 steps, significant additional computation is performed by radiation calculations.

A multitasking version of an application requires more main memory than its singletasking equivalent. Given (a) the desire to maximize the resolution and (b) the shortage of main memory, it is important to select a multitasking strategy which has low memory requirements.

It turns out to be convenient and efficient in memory to split Scan 1 and perform it in 2 pairs of subtasks with a synchronizing point in between. This is because each northern row generates the symmetric part of a Fourier component, while the equivalent antisymmetric part is generated by the appropriate southern row. Both components are combined in different ways to provide contributions to the legendre transform. By computing one northern row and one southern row simultaneously, not only is the memory requirement minimized, but also the legendre computation is performed efficiently.

Part of the diffusion calculation is also multi-tasked and Scan 2 can be computed 2 rows at a time (see Figure 2).

There remain some relatively small parts of the code which are computed in singletasking mode.

The memory requirements for this multi-tasking strategy are 1.8 Mwords. Note that alternative strategies are, of course, possible. However, subtask structures which may be preferred for optimizing reasons require either more central memory or additional SSD.

TIMINGS

All the timings reported here are elapsed times corresponding either to a single time step or to a complete 10-day forecast.

For a normal timestep:

singletasking:	19.73 seconds/step
multitasking:	11.36 seconds/step
speedup ratio:	1.75

These times correspond to a total time of 5 hours 15 minutes for a 10-day forecast, including the creation and post-processing of history data.

Since the above timings are very simple and made at the very highest level, they tell nothing about the behavior of individual tasks within the model. Currently, there is no support within the Cray multi-tasking library for obtaining detailed timings. Consequently, all the following timings were obtained by inserting code into the model at strategic places in order to record times as reported by the real time clock. The measurements were done in such a way as to disturb the model as little as possible. The model was run in a dedicated environment with no disturbances other than any caused by the operating system (COS 1.13). Analysis of the measurements was done subsequently in a normal batch environment.

The average times taken by each of the tasks as identified in the previous section are shown in Figure 3.

By measuring the time taken by the Cray multi-tasking library routines, it is possible to obtain estimates of the cost of starting tasks, etc.

For TSKSTART, three distinctly different times are observed as follows:

40 milliseconds	for one case only
0.4 milliseconds	for 96% of all TSKSTARTs
0.04 milliseconds	for 4% of all TSKSTARTs

The expensive start corresponds to the very first TSKSTART in the complete application, when additional memory has to be requested from the operating system for table space.

The intermediate time corresponds to the case when a 'logical CP' has to be connected to a 'physical CP'.

The shortest time corresponds to the case when a physical CP is already connected. In this execution, the Cray multi-tasking scheduler has released the physical CP in nearly all cases before the next task is created. The small percentage of fast TSKSTART times were all observed for PROCESS 2, where there is a very small time gap after completion of PROCESS 1.

By tuning the actions of the library scheduler (CALL TSKTUNE), it is possible to modify this behavior so that a terminating task retains connection to a physical CP, allowing the cheapest TSKSTART time when the next task commences. This is a valid strategy for a dedicated environment and allows 90% of the TSKSTART costs to be only 40 microseconds.

The measured minimum times for other multi-tasking calls are:

TSKWAIT	60 microseconds
LOCKON/LOCKOFF	1.5 microseconds

The approximate total overhead is 82 ms per time step (0.7%).

An obvious conclusion is that task overheads are small compared to the size of tasks which exist in the spectral model.

INEFFICIENCIES

By measuring the amount of time spent outside of the tasks, it can be seen how much of the code has been multi-tasked and therefore what additional improvements might be made in the future (see Figure 5).

The TSKWAIT time reported in the previous section was the minimum observed, i.e. for the case where the master task completed after the started task

and was therefore not held up in the synchronizing process. By examining average TSKWAIT times, it is possible to obtain estimates of how imbalanced the pairs of tasks are. Figure 5 shows that these imbalances account for nearly 4% of the overall model time. Most of the imbalance was observed in PROCESS 1. PROCESS 2 and PROCESS 3 imbalances were smaller by a factor of 9.

There are at least 2 reasons for this imbalance. One concerns LOCKS and will be discussed below. The other concerns the nature of the computation in grid-point space (part of PROCESS 1). Although the amount of work done for each latitude line is exactly equal for the dynamics part of the code, this is not always true in parts of the physical parameterization. Convection and condensation calculations are affected by synoptic conditions and will therefore vary in space and time. The magnitude of these variations in terms of computing expense has not yet been measured.

LOCKS are used to protect critical regions of code in some 20 places, mostly for statistic gathering purposes. These locks all occur in PROCESS 1 and are mostly insignificant in time. However, some random I/O is carried out to a single dataset which is common to both tasks and in the current Cray software, a lock is applied whenever I/O is initiated to any dataset. Indications are that this causes most of the imbalance observed in PROCESS 1.

EXECUTION ON A CRAY-X48

It is a straightforward process to extend the strategy to utilize 4 processors. A second north/south pair of lines of latitude are processed simultaneously, and the only new problem arises in the direct Legendre transform, where every northern row adds a contribution into one half of the spectral array and every southern row updates the other half. To avoid 2 rows updating the same elements of the spectral array simultaneously, some locks are necessary, but for efficiency reasons their effect must be minimized. Currently this is achieved by splitting the work domain into 4 pieces and by making a dynamic decision as to which piece to perform next using the LOCKTST function.

The measured performance on a CRAY-X48 is as follows:

processors	1	2	4
elapsed seconds/time step	19.3	10.3	5.5
speedup	-	1.87	3.5

These timings lead to a predicted overall cost for a 10-day forecast in an operational environment of 2 hours, 20 minutes.

Comparison with the performance on the X22 (Section 7) shows a small difference for the single processors execution due to a faster SSD channel speed. The much larger difference for the dual

processor execution is due to insufficient memory banks on the X22, where the CP speed is retarded by an average of nearly 10%.

FUTURE DEVELOPMENT

The existing 4-processor version of the model provides the basis for acceptable execution on an X48. However, the static nature of the task balancing leads to inefficiencies which can be largely removed by changing to a dynamic strategy. At a cost of increased memory requirement, it should be possible to reduce execution time by another 10%.

GENERAL STRUCTURE

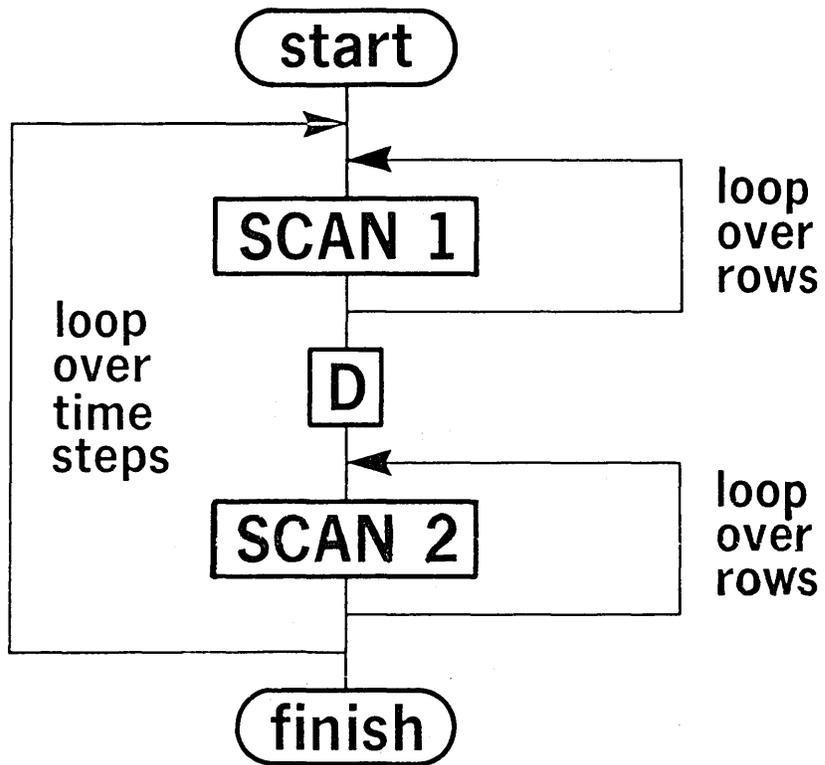


Figure 1

MULTI-TASKING STRUCTURE

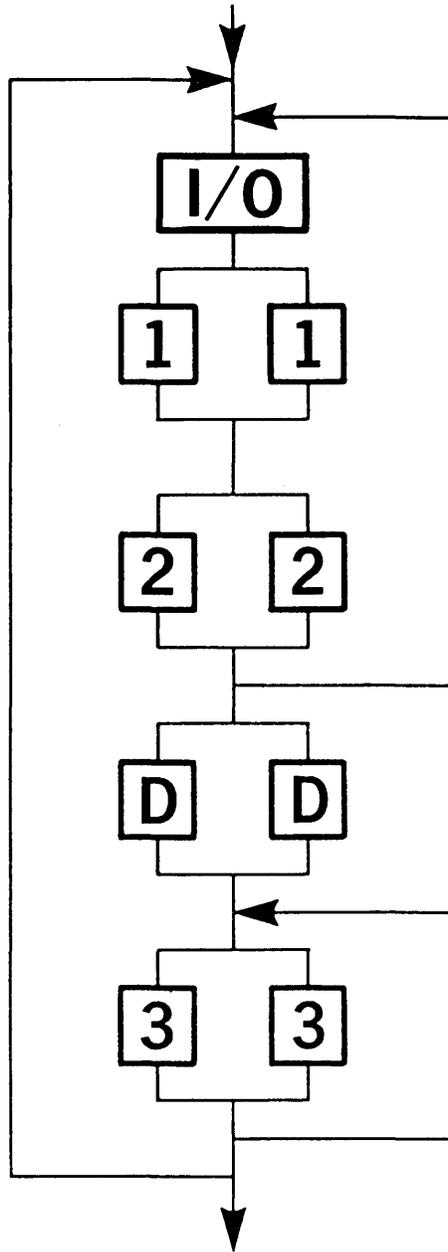


Figure 2

PROCESS TIMES

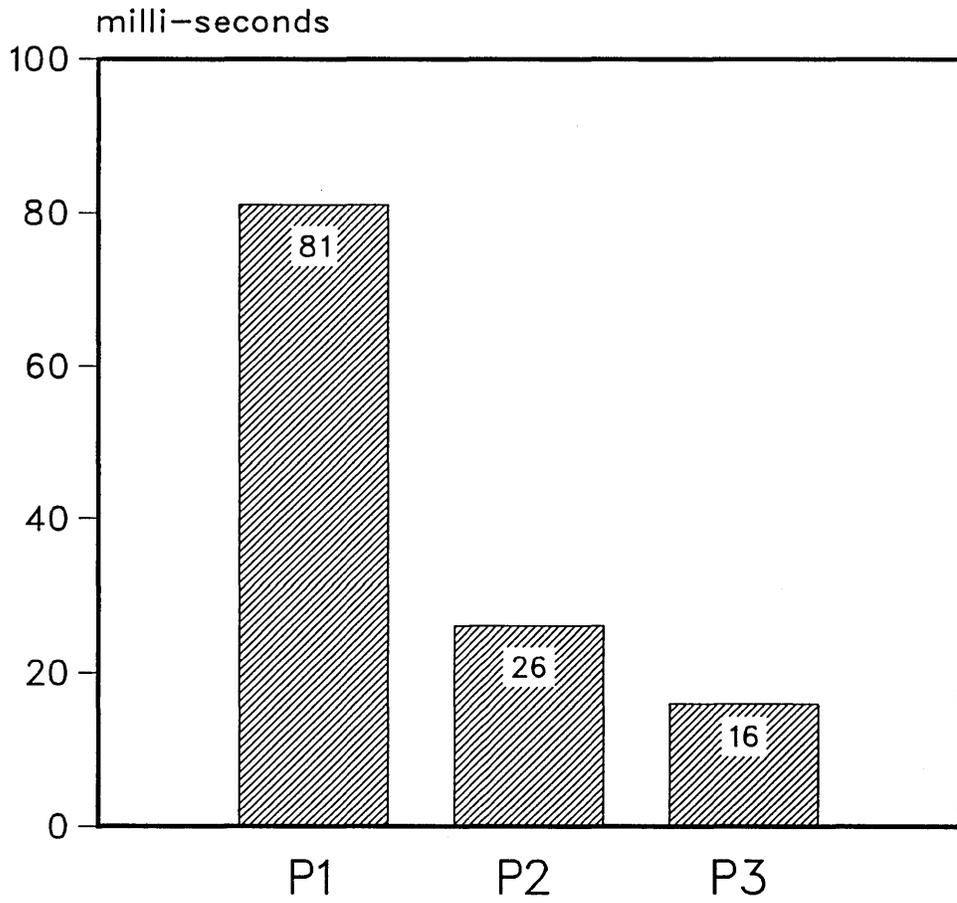


Figure 3



TSKSTART COSTS

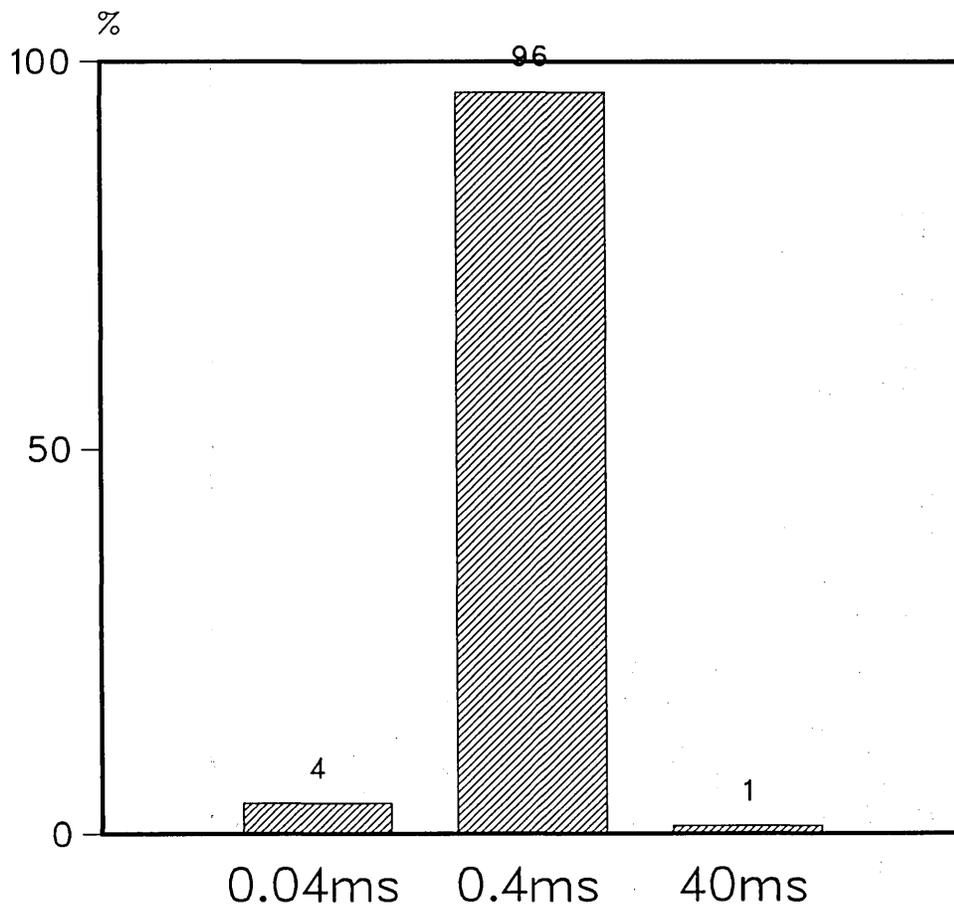


Figure 4



MULTI-TASKING EFFICIENCY

2 processor model

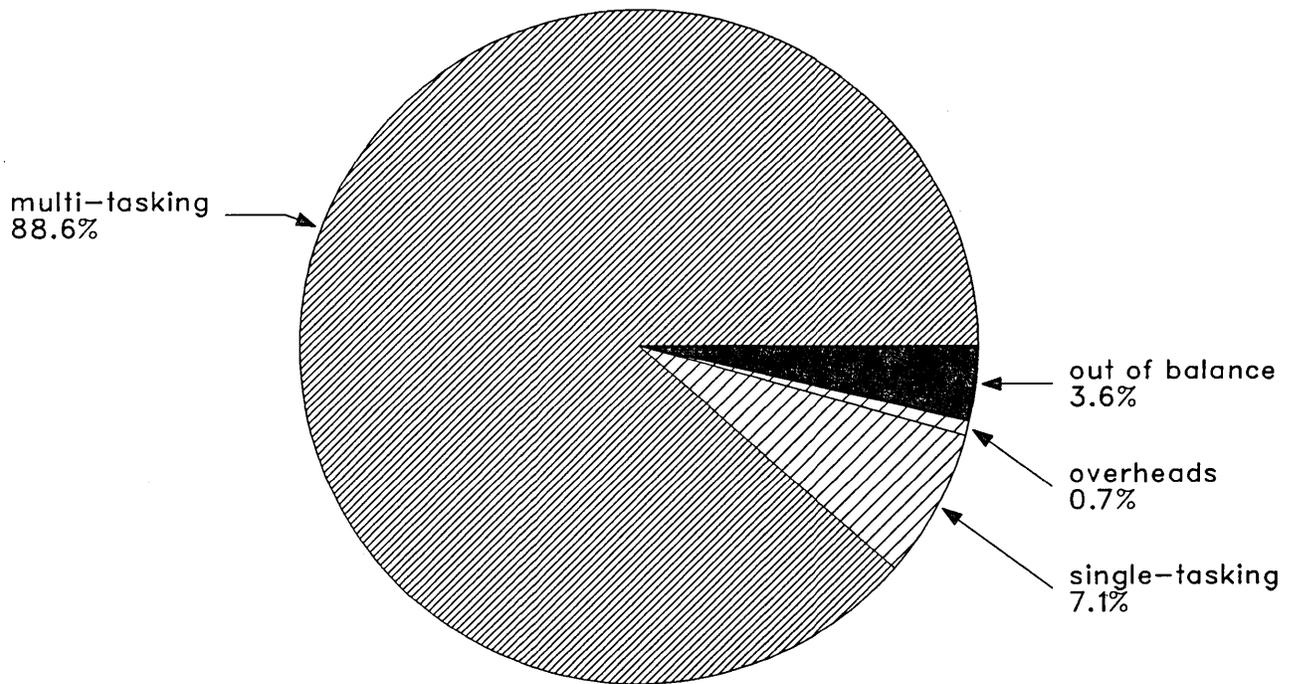


Figure 5



CMTS - A CRAY MULTITASKING SIMULATOR

J.D.A. David

CEA, Centre d'Etudes de Limeil-Valenton
Villeneuve-St-Georges, France

ABSTRACT

CMTS is a program in binary library that can be used on CRAY-1 and X systems, and on CDC CYBER systems, to simulate the CRAY Multitasking library. No source change is required to run programs with either Cray 'Multi' library, or CMTS. CMTS allows checking the correctness of multitasked programs to be run, and gathering statistics about processor use, for instance, the speed-up one can get from algorithms using several processors simultaneously. How CMTS works and its internals will be described, and examples will be given of use on real programs with comparison to real X-MP/48 benchmarks.

INTRODUCTION

Why CMTS

At the time we started to write CMTS, end of 1983, first specifications of Cray Multitasking were just known, and we wanted to be prepared to benchmark X-MP/24 and later X-MP/48. So we wanted to first be sure that future benchmarks would be correct with respect to Fortran and Multitasking Library syntax (so that benchmarks would run as soon as possible on X-MP). Second, we wanted to experiment multitasked algorithms, test them with real results and real 'multitasked execution', and then evaluate their performance and get a rough evaluation of the expected speed-up and if necessary, tune or modify algorithms. Another reason was to debug codes at ease in France, and check for hidden bugs that would lead to deadlocks or unused and/or over-used branches of program (e.g., due to a misconception, some event chain would never occur and a task never be activated).

What is CMTS?

CMTS was conceived to emulate the standard Cray Multitasking Library, so the user would have to do minimum changes to programs and/or JCL to use CMTS rather than standard Multi. Another objective was to provide the user with an effective assistance to debugging, at all levels - that is, from verifying that calls were done with acceptable values of arguments, to catching deadlocks, and including diagnostics of probably bad use of

resources (locks/events) and a trace facility for 'task stepping'.

CMTS was primarily conceived to run on the CRAY-1/S that Limeil center had at that time, but a by-product was that it can also run on Cyber system (with NOS/BE). The Cyber was used as a front-end for the Cray, and interactive debugging of CMTS was then made possible for faster development. CMTS has also been implemented and tested on a Cyber 76 (SCOPE 2) that was also available.

CMTS DESIGN

The primary requirement for CMTS definition was to emulate all public calls to Cray Multitasking Library. To this we added some more calls either as an implementation convenience or as a possible user convenience - for example, a function to give current task ID was added (and is heavily used in CMTS), and a subroutine MTRCLL that does 'nothing' but permits the user to get better simulations, and especially to emulate busy waitings (that would be otherwise impossible to simulate on a 1-CPU mainframe without access to timer-counter interrupt). Also, to implement CMTS on CYBER system, which does limit Fortran names to 7 characters (CRAY limit is 8), we decided to use all 6 character names, beginning with the prefix MT. A secondary prefix (TK, LK, EV, I (for Internal) specifies the category of the routine. Only functions callable by user have a different convention; that is, first letter is compatible with Fortran implicit typing, and the name ends with MT. On Cray, a set of stub subroutines with Cray Multitasking Library names calling CMTS MTXXXX routines enables user to have the standard interface.

The main difference with Cray standard is that CMTS does not know the 'task common' notion. First, this notion was added to Cray primitives after the start of CMTS. Also, there were some problems with it - the semantics was not clear (did a child task inherit a copy of the task commons of its mother? to what value task common is initialized?). It was not standard Fortran compatible and then either needs a pre-compiler or user source code modification. For these reasons, and also because it was not easy to

implement in CMTS while user could easily emulate it with standard commons and indexing from task ID or value, Task Common was dropped from CMTS.

CMTS PHILOSOPHY

As CMTS runs on 1-CPU mainframes, it cannot emulate tasks simultaneously, so it emulates them in turn, with a (reverse by default) round robin algorithm. Task switch occurs inside CMTS library routines when they are called, if current task cannot proceed (default), or at user option, when a 'time-slice' is elapsed. As CMTS cannot choose times when it is called (in fact, this time-slice is a 'minimum' value, and doesn't suppress all timings problems - for example, without calls to MTRCLL), CMTS cannot catch busy waits inside tight loops.

Timing information was appended as a second thought, as it can give invaluable information about algorithm performance (primary objective was to be able to run multitasked programs, and effectively verify that there were no evident bug). Timing is done by CMTS keeping 'clocks' for CPUs and tasks, but it is not used otherwise (except for task choice for free CPU assignment). What it means is that CMTS processes events (LOCK ON/OFF, EVPOST/WAIT/CLEAR, TASK START/WAIT/END) in the order it encounters them in the course of the simulation, not in the order in which they would occur in a real multitasked system with the simulated number of CPUs. This can lead to wrong timings, and it could also lead to wrong results for algorithms which were time-dependent (but these algorithms would likely be indeterministic). In fact, CMTS execution corresponds to one on 1-CPU computer, even if it computes the times the same sequence of events occurring on n-CPU's would take. The most important fact remains that if CMTS declares an algorithm wrong, it is almost certainly wrong, and that timings that CMTS gets for most event-driven programs are quite accurate (see Benchmark comparison part).

CMTS INTERNAL DESIGN

Tasks are handled in a straightforward manner. Starting a task consists of forking the current task (UNIX fork call), then calling the associated subroutine, then ending the current (child) task (UNIX exit() call). Wait for end of task is handled by a list (chained) of waiting tasks.

Locks have two notable particularities. First, in early design it was thought that locks were in fact critical sections, although this was corrected later. From that remains the notion of 'ownership' of lock (the owner is the task that locked-on the lock), and the diagnostics (optionally fatal) that warn user from inter-task lockon/lockoff. Second, the tasks waiting for a lock are managed in a FIFO way. ('First' in emulation, not in emulated time order) - but an exception was introduced for speeding-up emulations. When a task does frequent, but short,

exclusive accesses to shared data (may be for updating purposes), CMTS prefers to keep temporarily ownership of lock to this task while it is emulating it. It means that the current task has a 'short circuit' access to locks while it is emulated in the real CPU. This does not modify timings if lockings are short, and it suppresses many costly task switches.

Logical CPUs were added with timings, to have timing computed for different CPU numbers. As timings, it does not introduce changes in emulation, except in order of choice of tasks during the round-robin of emulation. If there are more tasks active than there are available CPUs, some tasks will be put in waiting state, and then will be emulated later on, when a CPU becomes free because of a task blocking. The logical CPUs allocation algorithm is the simpler one. It is 'first' (from tasks waiting a CPU, the task having the oldest 'real' time) come, first served, without any priority; any blocking forces the task to release the CPU.

CMTS uses IDs to communicate with users. These ID are variables set to 'unique' (for a CMTS run) value, depending on ID type. Task IDs start at 50000, locks IDs start at 60000, events IDs start at 70000. This enables CMTS and users to check quickly ID validity.

HOW CMTS WORKS

CMTS can be thought of as a two-level modular structure. The first level is user callable, and does all bookkeeping about timings, statistics, and managing of task queue/locks/events states. It submits all the real task management to second level, via calls that create, delete, (re)start, and stop tasks. The second level takes tasks status and manages logical CPUs so that at most, nCPU tasks are enabled. Others are either waiting a task, a lock, an event (first level management), or waiting a CPU (second level management). It then takes tasks associated to logical CPUs, and runs them in (reverse) round-robin order. The scheduler selects tasks according to forced switch or time-slice switch as described above.

The swapping routine does swapping by writing task memory image to file by way of Fortran binary i/o. One routine, MTEXEC, is the executive and is always called each time an event that could switch to another task occurs. It is the 'system exchange processor'. The routine MTRCLL (which calls MTEXEC) does the same thing, but forces the round-robin to go one step and execute another task if there is one ready - this enables CMTS to emulate busy-waiting. The swapper routine is called only from MTEXEC, and in MTEXEC in branches of only one block-if, as last statement of the branch, so that in any case, it returns in MTEXEC at the same point. This is necessary because MTEXEC is swapped with task, as it contains in its context (local data) the (future) return point for the task.

CMTS IMPLEMENTATION

CMTS consists of 100 modules, of which 2 are CAL and 2 are COMPASS. The 'COMPILE' (or \$CPL) file contains about 7000 lines, giving 4500 statements.

All CRAY Standard Multitasking routines are supported, the only exception being the TASK COMMON notion. CMTS allows 33 logical CPUs, 33 tasks, 100 locks, 100 events to be simultaneously used. At most, 500 contiguous Commons areas (and 500 contiguous local areas) can be used. Fortran input/output logical numbers used are 98 for load map file ZZZZMP, 99 for options file MTOPTS, and 60 to 60+maxCPU for tasks swap files.

CMTS options, specified either in free/keyword format on file MTOPTS (read upon CMTS initialization - its use is to be able to run the same program with different number of CPUs without any change or recompilation), or as arguments in MTOPT call, allow user to specify CPU number, clock tick (for RTC emulation), message level, trace file, reprieve or end processing, and warning/fatal level for dubious locks/events usages (such as task 1 lockon/task 2 lockoff, lock released while on, event posted while it is already posted, etc....).

Output from CMTS is labeled with prefix identification containing logical CPU number, CPU/task real time (as computed by CMTS), task ID, task cp time. If required, trace file contains all events which did change status of any task, with all IDs specified; optionally, all calls to multitasking lib (CMTS lib) can be traced. Statistics are supplied at end of job step, with count/max/min of locks/events/tasks activity. Also, a message signals each task start (and each start of multitasking activity), and each task end with an estimation of task efficiency (real-time/CPU-time); at each multi-activity end, CMTS gives a rough estimation of the speed-up for the multitasked part just ended.

CMTS UTILIZATION - EXAMPLES

CMTS was used at Limeil to debug and evaluate 2 vectorized, multitasked codes that were run later on Mendota Heights CRAY X-MP/48.

The first code was a Monte-Carlo Neutron Transport code [2]. Task synchronization was done either by TSKSTART/TSKWAIT, or EVPOST/EVWAIT. LOCKON/LOCKOFF was used for critical section around common data updating. CMTS found bugs such as argument passing, like:

```
DO 1 I=1, N
  X=expression(I)
  1 CALL TSKSTART (1TSK,SUB,X,I)
```

(X and I values received by instances of SUB will be 'random', depending on relative timings of loop and taskstarts).

Other bugs found were missing critical sections, RANF generator interference with tasking (we were forced to use it as a non-sharable resource, with lockon/ranset/ranf/ranget/lockoff calls). A pseudo bug was that for CMTS, the main program was swapped as other programs, and then data local to main program could not be transmitted to other tasks (this does not occur with Cray Standard Multitasking Library). So we had to put shared data in COMMON, which, in any case, is always good practice.

The second code was a set of versions of Preconditioned Conjugate Gradient algorithm [3], which has no intrinsic parallelism (contrary to preceding algorithm), and has a very small granularity. Synchronization was done by means of locks, or of events, or by higher level routines (using locks and events) like SYNC (rendez-vous or barrier routine), and 'tokens' (dataflow (or Petri nets) approach - each task receiving and giving tokens for synchro). CMTS found deadlocks not foreseen, and signalled lost eposts that pointed that synchro was not done as intended.

Real benchmarks [4] enable us to compare Standard Cray multitasking results with CMTS provisions. Numerical results from algorithms were identical, except in cases of indeterministic runs (CMTS always gave the same value (for fixed number of processors), as user didn't change parameters, but X-MP gave 3 different results). Timings observed were equal to those predicted within 10% for 2 processors, and for 4 processors the difference can be observed (and verified from other measurements) to come from memory contention (timings degraded about 10-20%).

Example with algorithms INV synchronized with SYNC on 2 and 4 processors.

Number of CPUs	Theor.	Real	CMTS
	Sp-up	Sp-up	Sp-up
2	1.85	1.85	1.91
4	3.23	2.75	3.48
	on X-MP/48		on 1-S

CONCLUSION

CMTS is a powerful tool to test, debug and evaluate multitasked algorithms. Its debug options and statistics give the user invaluable information. Its predictions, although rather crude, are quite accurate to evaluate algorithms. Most of all, it doesn't need a real multitasking machine - it can even run on non-Cray systems.

CMTS is now included in CRAY BENCHLIB. It could also be ported, with minimal effort, to other mainframes for users that would like to test Cray multitasking.

REFERENCES

- [1] David, J.D.A. and Meurant, G.A., "CMTS User's Guide", CEA Report CEA-N-2432.
- [2] Chauvet, Y., "Cray Channels", Vol. 6, No. 3 (1984), Computer Physics Communications, Vol. 37 (1985), to appear.
- [3] Meurant, G., "Preconditioned Conjugate Gradient", LBL - 18023 (1984), BIT, Vol. 24 (1984), pp. 623-633.
- [4] Chauvet, Y., David, J., and Meurant, G., "Experiences Numeriques sur le CRAY X-MP/48", CEA Report CEA-N-2446, June 1985.

MULTITASKING

Chaired by
Margaret L. Simmons

THE MULTI-LEVEL DATA BUS APPROACH TO MULTITASKING

J. L. Owens

Lawrence Livermore National Laboratory
Livermore, California

ABSTRACT

This paper describes a method of program design that involves the separation of the data access functions from the operations on the data within a program. This separation allows the program developer to easily adjust the amount of parallelism and size of granularity of the resulting multi-tasked program. With this approach the programmer can, over time, move to finer and finer grain tasking and achieve a balance between the granularity and overhead.

INTRODUCTION

In the process of constructing multi-tasked programs we have observed that multi-tasking a program becomes a problem in data management. The program designer must make sure that the data values are available when a task needs them and that no other task has simultaneous access to those data values unless they are protected by some type of synchronization method. Fortran's pass-by-reference method of argument passing causes problems when a program is multi-tasked. The problem that is caused by passing by reference can best be illustrated by an example. Consider the following program segment:

```
DO 100 M = 1, 10  
CALL CALC(M)  
100 CONTINUE
```

If we assume that CALC uses M to select independent data areas to modify, the natural way to multi-task this routine is to change it to something like:

```
DO 100 M = 1, 10  
CALL TSKSTART(TID(M),CALC,M)  
100 CONTINUE
```

Although this starts up 10 tasks, programming it this way in Fortran leads to a bug. Depending upon how the tasks are executed, it is possible that all of the tasks started will see an argument of 10 for the value of M. This occurs because the location in memory holding the value of M is being changed as the tasks using that value are running.

This type of problem could be solved by giving Fortran the ability to pass by value, but it reappears when data is communicated via COMMON. Since data communication via COMMON is often used when large data sets must be communicated to many routines, we will need some method of controlling the access to variables contained in COMMON blocks.

In this paper, we describe a method of data access control that is analogous to the way data access is controlled within most modern computers. This method will allow us to communicate to independently running tasks via variables contained in COMMON but still make sure that the tasks each get a private copy of the data that they will modify. Most modern computers use a bus structure to control the movement of data between memory and the CPU. Boards are designed to plug into the bus and thus are able to obtain access to the data in the memory. Various control methods are used to control access to the bus. It is necessary for the boards on a common bus to each obey a bus protocol so that each board will know when it can have access to the bus. In the following sections of this paper we will describe a method of passing data to routines that functions much like the bus of a modern computer. The tasks will get subset selection values from a COMMON block. All tasks using the same bus will use information from this COMMON block, but they will each get different subsets to operate upon since they will select the subset selection values based upon a bus level number that was passed to them when they were started. It is the responsibility of the bus control routines to make sure that the subsets of data processed cover the entire data base that needs to be processed.

DATA COMMUNICATION OVER A MULTI-LEVEL DATA BUS

Before we describe the data bus approach to passing data into routines we will digress for a moment and consider how data is currently passed into and out of data computation centers. Consider the following fragment of Fortran code.

```
A = B + C * D  
E = F + G  
H = B + D
```

Traditionally we have thought of each of these statements as being performed one at a time but why not let all three of these statements execute at once on a multiple cpu machine. Since the data used and the values calculated are independent for these equations we could do that. However if we consider the following fragment of Fortran code we see why some control method is required.

```
A = B + C * D
E = A + F
G = A + E
```

In case of the equations above we see that before the second equation can run we must have completed equation one and before equation three can execute we must have completed both Eqs. (1) and (2). There are many forms that the necessary control can take. At the level of single lines like these that are operating on scalar variables, we would probably just run all three lines of code on the same CPU and use the traditional Fortran sequential statement execution convention to make sure that the necessary control was present. However, what if the code fragment was as below?

```
DO 100 I = 1, 1000000
100 A(I) = B(I) + C(I) * D(I)
```

```
DO 200 I = 1, 1000000
200 E(I) = A(I) + F(I)
```

```
DO 300 I = 1, 1000000
300 G(I) = A(I) + E(I)
```

Now we could have a performance problem if we simply let one processor run all three statements since it could take a long time to do each one. Also, if we note that the first element of the second equation can be calculated as soon as the first element of the first equation is completed, we see that a considerable amount of potentially overlapable execution can be lost if we run on only one CPU. Even if we let several CPUs process a given line and wait until all CPUs have finished a line before going on to the next line we still lose some of the available parallelism. When we consider that many modern machines have vector instructions that perform at 10 or more times their scalar instruction counterparts, we begin to see the problems in trying to apply as much as possible of a multi-processor to the above problem. One approach to this problem that we have been considering is to think of the data access control problems as if they were really bus access and control problems. The concept is to communicate between calculation centers by letting the centers themselves get their data from a software simulated bus with access to the bus controlled by event flags. While the bus concept provides some structure to the data flow control problem, it also leads to another problem. The bus itself can become a bottleneck! So we are led to the concept of a data bus that has many levels upon which data can be moved. By moving data in parallel over each of these buses we achieve a high level of parallel operation, and if we only pass over the bus the information to tell the tasks what subsets of data a given

task is to operate upon, we minimize the amount of information that must be passed. Also by defining the bus and the data it can access we get a modular method of construction so that subroutines and buses can be shared between several codes that do the same kinds of operations on the same data structures. The multi-level data bus approach can fit in well with a structured and modular method of program construction. However we do give up one of the favored concepts of structured design for those subroutines that are multi-tasked -- the concept of passing all data via arguments. Since we want the subroutines to operate as efficiently as possible we do not want to have to pay the indirect addressing, data copying, and space overhead that comes with passing arguments by value. But we can replace the argument passing mechanism with the multi-level data bus passing mechanism and retain many of the advantages of passing data via arguments.

The concept then is that the above program fragment would function as shown below:

```
(Wait until given access to a bus level via the raising of an event flag)
(Take from the appropriate bus level M and N)
DO 100 I = M,N
100 A(I) = B(I) + C(I) * D(I)
```

```
(Wait until given access to a bus level via the raising of an event flag)
(Take from the appropriate bus level M and N)
DO 200 I = M,N
200 E(I) = A(I) + F(I)
```

```
(Wait until given access to a bus level via the raising of an event flag)
(Take from the appropriate bus level M and N)
DO 300 I = M,N
300 G(I) = A(I) + E(I)
```

It should be noted that three different multi-level data buses are being used to control the data access in the above code fragment. First there is a data bus that controls read access to B, C and D and write access to A; then there is a data bus that controls read access to A and F and write access to E. Finally there is a bus that controls read access to A and E and write access to G. Note also that if the tasking and event handling overhead permitted it, we could start up many, many tasks that could all run in parallel (i.e., M - N could be small). It should also be noted that I, M, and N are local to each task so that in theory the bus control software could even allocate different amounts of work to different tasks. Since each of these computation centers is working from different values of M and N they can actually go back to the bus to get more data to work on when they finish the work they are currently doing. Thus the coding would really look something like:

```

50 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 100 I = M,N
100 A(I) = B(I) + C(I) * D(I)
   GO TO 50

150 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 200 I = M,N
200 E(I) = A(I) + F(I)
   GO TO 150

250 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 300 I = M,N
300 G(I) = A(I) + E(I)
   GO TO 250

```

In this form we can actually get along with a smaller number of tasks since each task gets more work when it finishes the work it is currently doing. However the designer of a program may wish to design the code so that each task is given the same amount of work as each of that task's clones and such that only one bus transaction is all that is necessary to accomplish the work. This will keep the communication overhead over the data bus to a minimum and minimize the number of event flags used. In the form above we have assumed that a given task keeps the bus it is using busy until it has completed the work assigned to it. Actually a task could release a given bus level after getting its value for M and N from the bus but we choose not to allow this since the reuse of a given bus level by other tasks complicates the bus control algorithms. However we have omitted another requirement of the bus control logic that cannot be left out if the bus is to function. This is the requirement for the initial assignment to each task of its bus level.

The bus that a given fragment of code uses is decided by the code fragment's data access requirements, but its bus level is decided upon dynamically and depends on when the task was started. The allocation of bus level then becomes a problem because each task must know its bus level in order to get its data, but at the startup time of the tasks these levels have not necessarily been decided upon. Another point we have left out is any discussion of the bus controller software that must not only control access to a given bus but must also work with other bus controller software to implement the control logic necessary for the correct operation of the program. In the next section we discuss the requirements for these software bus controller modules.

THE MULTI-LEVEL DATA BUS CONTROL LOGIC

Up until now we have not described how the tasks are started or how the control of a bus is carried out. This is because there are many ways that the control of a bus can be implemented. Below we will describe one of the ways the control of a bus could be implemented. At one extreme, consider a code that contains only one multi-level data bus. In this case at the beginning of the code the tasks that were to process data in parallel could all be started and as they are started they could be passed their level on the data bus. Since we are implementing multi-level data

buses which have only one task per bus level we could then have the actual task's code look much like that listed below. Even if there were many data buses in a given program we could still do the bus level allocation at task startup time if we have a different level counter for each multi-level data bus. If this approach is used then the actual programming might look something like the following,

```

(Get bus level number)
50 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 100 I = M,N
100 A(I) = B(I) + C(I) * D(I)
   GO TO 50

(Get bus level number)
150 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 200 I = M,N
200 E(I) = A(I) + F(I)
   GO TO 150

(Get bus level number)
250 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 300 I = M,N
300 G(I) = A(I) + E(I)
   GO TO 250

```

where the process of getting a bus level number would be simply to accept it as an argument at the time of task startup. All the tasks could be started at the start of the program and from then on use the same bus level for all data communication control. An alternative to this method would be to allow a task at the start of its run to ask for a bus level from the bus control routine via using some communication convention that would allow many tasks to each get a unique value for their bus level. A fetch-and-add primitive operation on a unique location for each bus could be used for this purpose. Of course other critical code section locking methods could also be used to accomplish the same purpose. For our purposes here we will assume that the simple allocation of bus level at task start time is all that is needed. This process might look much like that listed below.

```

(AT start of program, start the bus controllers which in turn start up the tasks
that will process the data under the control of the bus)

(Get bus level number)
50 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 100 I = M,N
100 A(I) = B(I) + C(I) * D(I)
   (Set the bus event to say that the data subset has been processed)
   GO TO 50

(Get bus level number)
150 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 200 I = M,N
200 E(I) = A(I) + F(I)
   (Take from the appropriate bus level M and N)
   GO TO 150

(Get bus level number)
250 (Wait until given access to a bus level via the raising of an event flag)
   (Take from the appropriate bus level M and N)
   DO 300 I = M,N
300 G(I) = A(I) + E(I)
   (Take from the appropriate bus level M and N)
   GO TO 250

```

Now we come to the bus control logic itself. This is implemented via one or more independent tasks for each data bus implemented. Logically we can look at the bus control logic as being required to do several things. First it must decide if it is time to allow data to move across the bus (i.e., it must find an answer to the question "Is this program ready to perform the calculation centers that are waiting for this bus")? Next it must select the subset of data and place it so that the tasks can access it. Then it must set the bus event to say that the data is on the bus and ready for use on the given bus level. It then must wait for the event that indicates the calculation has been completed and the output is on the bus. Then it must take data calculated and put it back where it belongs. Finally, if the bus is to be used again, it must wait until it is again time to put data on the bus for another round of calculation.

The job of the bus control logic sounds complicated but sometimes it can be greatly simplified. In fact if we consider the original scalar version of the equations that we had at the start and think about what happens in a slightly different way we can see how each of the functions performed by the bus logic gets done in normal programs. First think of the equations sitting there ready to do their calculations and the event that they are waiting for is the arrival of the program counter at their location. But before that event can happen another event occurs (i.e., the arrival of the program counter at another position) that allows some instructions to be executed to put the needed data in certain registers where the calculations assume the data will be when they are executed. Finally when the equations have completed their work another event (again the program counter arriving at a certain location) occurs that allows the output from the equation to be put back into the location that it needs to be in for future calculations (i.e., main memory). Of course in this simple case the bus logic gets mixed up with the equations themselves and we are proposing that it be explicitly separated from the equations. This separation will cause some extra overhead but if the amount of computation that must be done is large then the advantages of allowing the parallel execution of the program will more than make up for the extra logic of explicitly raising events to communicate between the bus control and the equations.

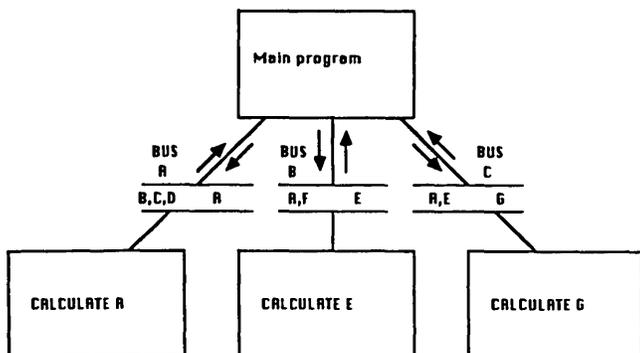
One advantage of the data bus approach is that it can hide from the equations the amount of work that must be done to place the data where the equations assume it will be. Thus the equations can assume that the data will be in a COMMON block and their coding simply refers to it via offsets passed to them over the bus (i.e., another COMMON block that is indexed by the bus level number that each task is given at its birth). On some machines the process of getting the data controlled by the bus may require the movement of the data from a large central memory to a smaller and faster local memory. Such data movement is hidden from the equations by the bus logic that takes care of such machine specific dependencies. This allows a program using the multi-level data bus design to be moved from machine to machine with machine

dependent changes being made only in the data bus control routines.

One thing that has not been spelled out is how the bus logic knows when it is time to cause activation of its particular bus. One method of exercising this control is to have another bus (a MASTER bus) that is really just an array of events with each multi-level data bus plugged in at a certain place (i.e., looking for a given event in the array). Then as certain points in the progression of the program execution are reached the appropriate events are enabled and the associated bus allows data transfer. When all the bus data has been moved over the bus and the output has been placed where it belongs, then the bus control can raise an event to indicate its job is completed for now and this could in fact be the event that started another bus to activate. Thus the buses themselves could accomplish what is done by the arrival of the program counter in normal programs (i.e., the control of the flow of the program's logic). This type of control may prove to be too restrictive and some interbus control may be needed that allows many buses to be active at any one time. In fact that is what is needed in the example given above so that the second and third equations above can run at the same time as the first one is working on its data. In this case the bus logic as described above can be carried out at the bus level number level rather than at the entire bus level and thus when a bus level has completed its processing it can cause a level in another bus to be activated. Since in theory the number of bus levels could approach the number of elements upon which the calculation is to be performed, any level of parallelism that is needed could in fact be implemented via this approach. Clearly at some granularity level the overhead will exceed the work done and we will have to be judicious in our use of very fine granularity.

As described above, the data bus approach appears to be much like the data flow approach. Indeed a data-flow-like execution can be implemented via this approach but so can a normal sequential program flow. We think that the most likely place for a data bus to be used is at a point where a procedure call is being made. If in changing a code into a multi-tasking code you are able to identify a procedure that will be multi-tasked then you have identified a place to consider the use of a multi-level data bus. It is just this ability to be used to change a totally serial program into a parallel program in a step-by-step manner that makes the multi-level data bus approach attractive. With this approach to transforming programs we think we will be able to move in a step by step and structured manner toward a more data-flow-like execution of programs. By identifying exactly what data is controlled by a given bus we will be able to maintain a better understanding of how the program is functioning. In fact, as can be seen below, the structure chart of a program being transformed into a multi-level data bus form looks much like the traditional structure chart with only the arguments being passed via a different approach. Note that in this approach the arguments to routines are usually passed via COMMON and only the subset selection is passed in the variables associated with the data bus. Thus this approach lets us see explicitly in the structure chart arguments that were

hidden in nondata bus programs because they were passed by COMMON. Below we show a structure chart for the code fragments we have used before in this paper. We now consider each fragment as a procedure since that is the way this approach will be implemented using the CRAY multi-tasking primitives.



If the data bus model of producing parallel programs proves to be a good paradigm then in the future it might be possible for compilers to replace argument passing at certain specified places in a program with a data bus type of data passing mechanism and thus automate the production of the data buses for a program. With machines such as the ETA machine where large blocks of data may have to move between local and remote memory, having the data bus type for arguments in the compiler could relieve the programmer from having to worry about the movement of the data.

CONCLUSION

We have described a method of controlling the flow of information into and out of subroutines that can be applied incrementally as needed to convert a Fortran program from a serial version to a multi-tasked version suitable for running on a multi-cpu computer system. Unlike other approaches that suggest that new languages may be needed or that additional features be added to the syntax of current languages, this approach simply removes a feature from Fortran for multitasked routines while leaving the nonmultitasked routines as they are. The feature not used is the argument passing by reference of Fortran. It is replaced by an argument passing method based upon selection of subsets of the data which are stored in COMMON blocks. The method allows finer and finer grains of parallelism to be used as incremental changes are made to a program. Ultimately a very data flow type of execution can be achieved.

EXPERIENCES WITH CRAY MULTITASKING

Eugene N. Miya and M. S. Johnson

AMES Research Center
Moffett Field, California

ABSTRACT

This paper covers the issues involved in modifying an existing code for multi-tasking. These include Cray extensions to FORTRAN, an examination of the application code under study, designing workable modifications, specific code modifications to the VAX and Cray versions, performance, and efficiency results. The finished product is a faster, fully synchronous, parallel version of the original program.

A "production" program, TWING, is partitioned by hand to run on two CPUs. TWING analyzes transonic airflow over a wing. Our preliminary study uses a greatly reduced data structure on a shared memory, multi-headed VAX. The program is then moved to a Cray X-MP/22 under version 1.13 of the Cray Operating System (COS) using Cray FORTRAN (CFT) versions 1.13 and 1.14.

TWING is a well-structured, highly vectorized program that runs on one processor. *Loop splitting* (performed manually) multi-tasks three key subroutines. Multi-task TWING uses two CPUs simultaneously in determining airflow above and below an airfoil; acting as two operator-independent flows. Another portion of the program splits processing into wingtip versus "surrounding" wing flows.

Simply dividing subroutine data and control structure down the middle of a subroutine is not safe. Simple division produces results that are inconsistent with uniprocessor runs. The safest way to partition the code is to transfer one block of loops at a time and check the results of each on a test case. Other issues include debugging and performance. Task startup and maintenance e.g., synchronization are potentially expensive.

Future research considerations involve the development and integration of a FORTRAN preprocessor for higher-level, explicit control of multi-tasking. Despite these problems, the partitioning of certain pre-existing programs looks promising.

Introduction and Outline

The search for improved performance has focused on using different forms of *parallelism* to achieve speed increases.¹ To this end, Cray Research, Inc. (CRI) introduced vector processing and, most recently, user-accessible multi-tasking (Larson, 1984; Research, 1985; Research, 1984). The Cray work on multi-tasking takes a "coarse grain" approach to parallelism in contrast to the "fine grain" parallelism of vector instruction sets or dataflow (Dennis, 1979). Multi-tasking was not introduced without tradeoffs such as this.

The issues raised with the introduction of multi-tasking and multiprocessing involve more than performance. Multi-task programs may require major changes in their algorithms: storage management, and code. Toward this end, new or modified programming languages are needed.

Explicitly parallel languages must handle problems beyond the scope of conventional programming languages. These issues include data protection, non-determinism, process management (i.e., creation, scheduling, deletion), interprocess communication, synchronization (i.e., deadlock and starvation), and error and exception handling (Denning, 1985). These problems are well documented with the Carnegie-Mellon's multiprocessor research (Jones, 1980). There are few simple solutions,² and tradeoffs must be made. Grit and McGraw compare parallel applications programming to operating systems programming in sheer difficulty (Grit, 1983) thus creating more trouble.

System timing must receive careful consideration in multi-task codes to avoid inconsistent results and deadlock. A sequential code hacking style is dangerous in this

¹The terminology is varied, colorful, and highly confusing. Among other phrases, we have: parallel processing, multiprocessing, polyprocessing, distributed computing, decentralized computing, and so forth. Each phrase has a slightly different meaning: enough to make communications difficult. CRI makes the subtle distinctions that *multiprogramming* means multiple jobs working on a CPU (e.g., time-sharing), *multiprocessing* means work done on multiple physical CPUs working multiple jobs (i.e., without regard for jobs), and *multi-tasking* means multiple physical CPUs working *cooperatively* on a single problem.

²Jones and Gehringer specifically classify distributed system issues into problems of 1) consistency, 2) deadlock,

environment. Care is required when dividing a problem into multiple tasks to avoid inconsistency. This division is called *partitioning* or *decomposition* as well as other terms.

Several partitioning schemes can execute codes in parallel (Jones, 1980). The most common are *pipelining*, *spatial partitioning* (by problem space or machine storage), or *relaxation* that removes assumptions of data consistency. David Kuck is best known for his research on automatic partitioning (Kuck, 1980). This paper covers the subject of partitioning an existing application program by hand.

The program "TWING" is the vehicle that we use to explore the issues surrounding multi-tasking. This report covers:

- Existing Languages: Issues and Problems
- The Cray Multi-tasking Implementation
- The TWING Program
- Modifications to TWING
 - The 2-Processor VAX Version
 - The 2-Processor Cray Version
- Debugging and Other Consequences
- Performance Issues
- Discussion and Conclusion

Our programming style is conservative and defensive. We assume the multi-task program will not execute the first time. We chose a synchronous algorithm and sought results identical to results using uni-task TWING. This work stresses the importance of careful analysis, design, and testing.

Existing FORTRAN Drawbacks

As background, it is useful to understand the problems inherent with standard FORTRAN and multi-tasking. FORTRAN is not currently designed for or intended to run in a parallel environment. New problems arise in multi-tasking such as synchronization, communication, error handling, and deadlock. An excellent survey of language issues and various attempts at solving them appears in *Computing Surveys* (Andrews, 1983).

First, the standard FORTRAN language lacks process-creation primitives and structures. The SUBROUTINE is the closest FORTRAN object resembling a process or a TASK. Second, the language lacks features for explicit synchronization and protection

3) starvation, and 4) exception handling.

such as semaphores (Dijkstra, 1968) (i.e., ALGOL-68), monitors (Hoare, 1974) (i.e., concurrent Pascal), or rendezvous (i.e., Ada³) (DOD, 1980). It also lacks explicit communication features such as mailboxes.

Each of the aforementioned synchronization features has assumptions of *atomicity* (uninterruptability) which is critical for maintaining a degree of consistency that standard FORTRAN cannot currently provide. Synchronization is a technique normally reserved for operating system programming (using libraries) since it offers "hazardous" user facilities.⁴

Lastly, the software engineering problems associated with FORTRAN are accentuated in a multi-tasking environment. These problems are documented elsewhere (Dijkstra, 1968): they include GO TOs and the lack of modern data structures. An example of these tradeoffs is the inability for Cray multi-tasking FORTRAN to coherently perform multiple RETURNS.

It is not easy to add these features to the FORTRAN language. These features conflict with existing language semantics. The programmer must locate and manage *side effects* on globally referenced memory (such as COMMON variables), call-by-reference parameter passing, and manufacturer-dependent features. These side effects also occur at the lower vector-processing level: Cray users have modified their programming style to accommodate them. We can similarly expect users to adopt a multi-tasking programming style.

Cray Multi-tasking FORTRAN extensions

The existing Cray Research supercomputer line performs efficiently by using a vector instruction set. Performance improvement is achieved by using regular data-access patterns on arrays and their indices. Currently, multi-tasking seeks to achieve performance improvement using multiple processing units.

Cray Research has a set of primitive extensions to support multi-tasking in version 1.13 of their CFT FORTRAN compiler

³Ada is a trademark of the Ada Joint Project Office of the US DOD.

⁴There exists the potential for user-induced system deadlock.

(Larson, 1984). These extensions currently allow several *virtual* CPUs to execute simultaneously on one to four *physical* CPUs. These primitives are invoked using subroutine CALLS. They are useful for creating more elaborate synchronization mechanisms such as monitors (Hoare, 1974).

The Cray primitives fall into three general categories:

- TASK creation and control
- EVENT creation and synchronization
- LOCK creation and protection

The primitives are controlled using three basic data structures: a TASK control array (INTEGER type containing two or three elements), EVENTS, and LOCKs (both of type INTEGER) all explicitly assigned (i.e., created).

An extremely important semantic⁵ difference is the handling of storage (primary memory) in this version of FORTRAN. Local storage in normal FORTRAN has a static allocation resulting in possible *side effects*.

The new multi-tasking CFT FORTRAN requires a dynamic or stack-based allocation of storage more characteristic of ALGOL-like languages such as Pascal or C. This is necessary for TASK creation and migration. Local storage (scalars or arrays) now has a finite lifetime and scope. A programmer cannot use a value left over from a previous subroutine CALL or assume values are initialized to zero (0). This is a radical departure from standard FORTRAN. The next four sections cover these primitives and their effects in greater detail.

TASK Control

We begin with TASK creation. A user controls a concurrent object called a *TASK* that is invoked like a SUBROUTINE. The TASK is defined like any other SUBROUTINE except that its name must explicitly appear in an EXTERNAL statement before a CALL, and its storage gets handled differently. The specific TASK syntax primitives are shown in figure 1 where SUBNAME is the SUBROUTINE name, and ITCA is an INTEGER *TASK control array*. Note,

⁵We mention this because there are no FORTRAN keywords (i.e., syntax) associated with this problem: it's semantic.

```
CALL TSKSTART(ITCA,SUBNAME,arguments!)
CALL TSKWAIT(ITCA)
```

Figure 1. Cray TASK primitives.

restricted, positional SUBROUTINE arguments are passable.

A *TASK control array* is a simple data structure that holds TASK control data for a scheduler that is loaded with the program on execution. This scheduler is distinct from the operating system's scheduler in that it governs user defined TASKs rather than JOBS.

The TASK is created using the TSKSTART call. TSKSTART is similar to a **fork** in languages like ALGOL-68 except a separate address space is created, much like a separate space for a FORTRAN subroutine. The effect is like a subroutine CALL with one major exception: subroutine CALLs are synchronous and consequently wait, unlike TSKSTART calls

The following program fragment (figure 2), listed in parallel, illustrates the creation of a TASK. Note that the subprogram allocating the TASK control array must not lose the TASK control array storage! Severe problems will result!

A "TSKWAIT" statement could force a crude explicit synchronization on execution of a RETURN statement within task A. The section on **Debugging** will touch on the usefulness of TSKWAIT. More refined

synchronization is available using EVENTS and LOCKs. There are also TSK calls covered in the Cray documentation that report TASK information or statistics (Research, 1985).

Cray support of multi-tasking includes a simple deadlock-detection mechanism. Deadlock occurs when all user TASKs are waiting for a condition that never occurs. This goes for synchronization using TSKWAIT, EVENTS, or LOCKs. Care is required, particularly in using EVENTS because these functions are not necessarily atomic (indivisible). Deadlock is discussed further in the section on **Debugging**.

EVENTs and LOCKs

Synchronization and consistency protection use combinations of EVENTS and LOCKs. Both are useful for simple synchronization. The key difference between an EVENT and a LOCK is that a LOCK forces tasks to run in a First-In, First-Out (FIFO) order. An EVENT is comparable to a "broadcast," and many TASKs can run at once. It is also important to clear or reset a LOCK or EVENT at appropriate times.

```
PROGRAM
INTEGER TA(2)
EXTERNAL A
...
CALL TSKSTART(TA,A,arguments)  SUBROUTINE A(parameters)
...
END                               END
```

Figure 2. An illustration of simple TASK creation.

EVENTS and LOCKS are created by using subroutine CALLS which assign special protection in the same manner in which TASKS are created. Basic arithmetic and logical operations are disabled for these objects until they are released. The specific primitive SUBROUTINE CALLS are:

EVENT Control	LOCK Control
EVASGN(IEVAR)	LOCKASGN(LCK)
EVPOST(IEVAR)	LOCKON(LCK)
EVWAIT(IEVAR)	LOCKOFF(LCK)
EVCLEAR(IEVAR)	LOCKREL(LCK)
EVREL(IEVAR)	

in which IEVAR and LCK are INTEGERS assigned as EVENTS or LOCKS. The following is a simple two-TASK synchronization using EVENTS in two separate executing TASKS. The scope is shown by the bounding boxes of figure 3. If an EVENT or a LOCK is CLEARED or RELEASED while some TASK is waiting, the consequences are nondeterministic and can be disastrous.

If combinations of EVENTS, LOCKS, and COMMON memory are used, it is possible to make more elaborate synchronization mechanisms such as semaphores and monitors. Sequential *critical sections* of code and data need protection using these synchronization primitives. Problems of inconsistent synchronization are covered in the next section.

Communications

Communication takes place through one of three mechanisms:

- CALL-by-Reference parameter passing
- Global COMMON memory

TASK COMMON memory

Data is passed using shared (e.g., COMMON) variables. This is the principal means of communication and requires care in use.

A TASK-local COMMON (e.g., TASK COMMON) is available in version 1.14 of the CFT compiler. It is similar to the more global COMMON except that its data is accessible only to objects (SUBROUTINES) within a particular TASK. Maintaining a consistent system state is a chore left to the user.

Consistency is threatened by three basic hazards. Suppose A and B are two TASKS running in parallel and sharing a variable V. The hazards are based on the order in which processes access V: a timing problem. The first hazard is the *read-write* hazard - having one TASK prematurely reading a stale value before the appropriate write. The next is the *write-read* hazard: having one TASK prematurely "clobbering" a value before it could be read. The last hazard is the *write-write* hazard in which one TASK writes over values that never get a chance to be read [particularly difficult to detect]⁶. The Cray is not responsible for these potential user errors of timing.

Storage and Subroutine Linkage

The actual handling of storage differs vastly from conventional static FORTRAN. This has its greatest effect on SUBROUTINE

⁶The memory on the Deneb/Heterogeneous Processor (HEP) is an attempt to solve this problem. If variables receive a special declaration, they are forced to alternate reads and writes using a unique semaphore memory system.

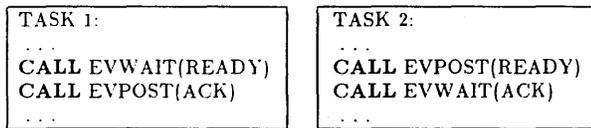


Figure 3. Synchronization of two TASKS using EVENT flags. Boxes represent different address spaces.

and FUNCTION linkages. The semantics of these new linkages prompt some users to name this an entirely different language (e.g., "not-FORTRAN"). Old memory-saving tricks such as statically defined and allocated variables left for a second subroutine CALL are now undefined and may contain unreliable data. Users cannot assume values are initialized to zero (0). Expressions in parameter lists involve similar problems.

Those readers familiar with dynamic storage management in scoped languages such as ALGOL, C, Pascal, or LISP should grasp these concepts easily. FORTRAN simply does not offer the protection mechanisms to ensure consistency of data in a multiprocess environment. The user must actively manage the data consistency and program defensively.

The Mathematical Basis for TWING

TWING is a program that solves the conservative full-potential equation, using a fully implicit, approximate-factorization algorithm. The program solves for stable state airflow over a wing flying at transonic velocity. TWING is the development of Dr. Terry Holst and Scott Thomas (Thomas, 1983) at the Applied Computational Aerodynamics Branch, NASA Ames Research Center.

Figure 4 is a schematic of the finite difference mesh over which the flow solver operates. From this representation in "physical space", the problem is transformed into a "computational space" [figure 5] which preserves the orthogonality of the mesh lines throughout the computational domain.

A mathematical representation of this flow solver is given in the derivation of equation 1.c. The three-dimensional, full potential equation (in x,y,z coordinates) is presented in equation 1.a. The transformation into computational coordinates (ξ, η, ζ coordinates) yields equation 1.b. In this equation, U, V, and W are terms composed of Φ_x , Φ_y , and Φ_z combined with assorted metric quantities. J represents the Jacobian of the transformation. The finite-difference approximation of this transformed equation (1.c) employs backward difference operators in the ξ, η , and ζ directions. This yields the finite-difference approximation in equation 1.c. The special density coefficients $\bar{\rho}$, $\bar{\rho}$, and $\hat{\rho}$ introduce an artificial viscosity term into the calculation. The residual term $L(\Phi)$ obtained from this equation is used in the first step of the factorization scheme outlined below.

An outline of the three-step approximate-factorization scheme is shown in the derivation of equation 2.c. In step one (equation 2.a), an intermediate term $G(i,j)$ is computed for each point on a given "k-shell" of the mesh by solving a tridiagonal linear system along each η line (i.e., $\xi = \text{a constant}$) extending from the symmetry plane out to the freestream sidewall. In step two (equation 2.b), $G(i,j)$ computes another intermediate term $F(i,j,k)$ for each point in the "k-shell." This step requires the solution of a tridiagonal linear system along each ξ line (i.e., constant η) extending from the upper vortex sheet around the leading edge to the lower vortex sheet (figure 6). Finally, when $F(i,j,k)$ has been computed for every point in the three-dimensional mesh, the correction factor

$$(\rho \Phi_x)_x + (\rho \Phi_y)_y + (\rho \Phi_z)_z = 0 \quad (1.a)$$

The three-dimensional full potential equation (x,y,z coordinates).

$$(\rho U / J)_\xi + (\rho V / J)_\eta + (\rho W / J)_\zeta = 0 \quad (1.b)$$

The full potential equation in computational space (ξ, η, ζ).

$$\bar{\delta}_\xi \left(\bar{\rho} U / J \right)_{i-\frac{1}{2},j,k} + \bar{\delta}_\eta \left(\bar{\rho} V / J \right)_{i,j-\frac{1}{2},k} + \bar{\delta}_\zeta \left(\hat{\rho} W / J \right)_{i,j,k+\frac{1}{2}} = 0 \quad (1.c)$$

The resultant finite-difference approximation.

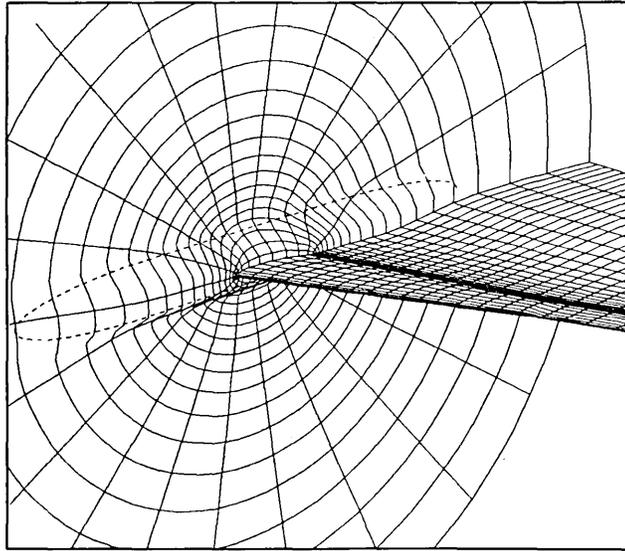


Figure 4. Sample finite difference mesh.

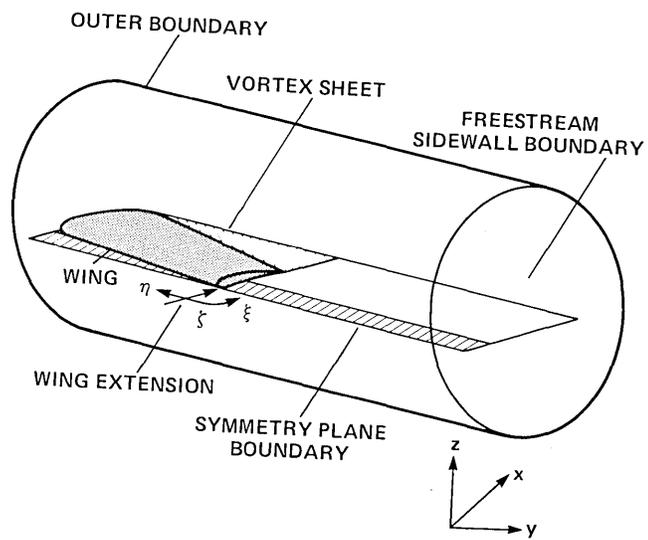


Figure 5. Transformation to computational space.

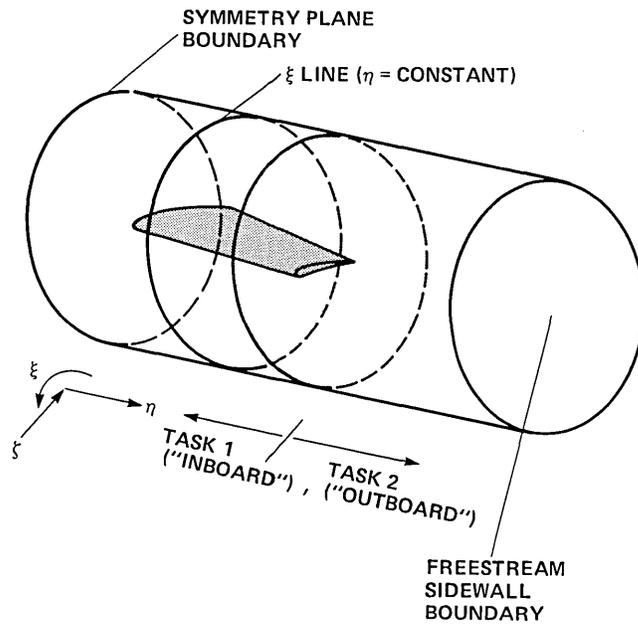


Figure 6. Computation divided into two tasks.

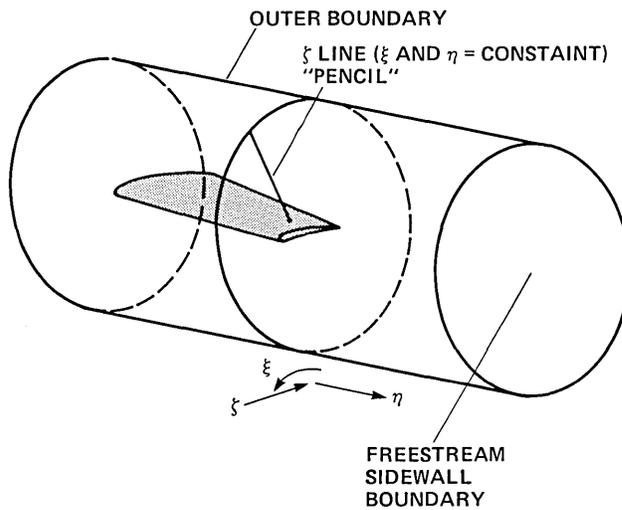


Figure 7. Computation done as a region of pencils.

Step 1:

$$\left(\alpha + \alpha \beta_{\eta} \left[\frac{V}{J} \right]_{i,j,k} \bar{\delta}_{\eta} - \frac{1}{A_k} - \bar{\delta}_{\eta} A_j \bar{\delta}_{\eta} \right) g^n_{i,j} = \alpha \omega L \Phi^n_{i,j,k} + \alpha A_{k+1} f^n_{i,j,k+1} \quad (2.a)$$

Step 2:

$$\left(A_k + \beta_{\xi} \bar{\delta}_{\xi} - \frac{1}{\alpha} \bar{\delta}_{\xi} A_i \bar{\delta}_{\xi} \right) f^n_{i,j,k} = g^n_{i,j} \quad (2.b)$$

Step 3: Correction factor C.

$$\left(\alpha + \bar{\delta}_{\xi} \right) C^n_{i,j,k} = f^n_{i,j,k} \quad (2.c)$$

Steps in the finite differencing scheme.

Program VTWING

```

Input subroutine (INPUT)
  READ mesh
  READ run-time parameters
Initialization subroutine (INIT)
  initialize the solution
  compute and store metrics
Flow Solver: (SOLVE)
  for each iteration do
    for each k-shell in mesh do
      get metrics
      compute density and density coefficients
      compute residuals
      solve for  $g^n_{i,j}$  and  $f^n_{i,j,k}$ 
    end k-loop
    calculate and apply  $C^n_{i,j,k}$ 
    output maximum residual and correction for iteration
    check convergence
  end iteration loop
output solution

```

Figure 8. Sequential structure of the TWING Program:

$C(i,j,k)$ is computed in step three (equation 2.c). This calculation proceeds from the outer boundary down to the wing surface, requiring the solution of a bidiagonal system for each ζ line (i.e., ξ and $\eta = \text{constants}$, figure 7) of the mesh. This correction factor is then added to the solution from the previous iteration, generating a new solution. This three-step process is repeated iteratively until convergence is achieved or a preset maximum iteration is reached.

An outline showing the code structure itself is presented in figure 8. The program first reads the physical coordinates of the finite difference mesh and its run-time parameters. The program then computes the metric quantities defining the transformation of the problem into "computational space" and writes these to disk.

At this point, the main iteration loop of the program begins. The program completes steps one and two (equations 2.a and 2.b) of

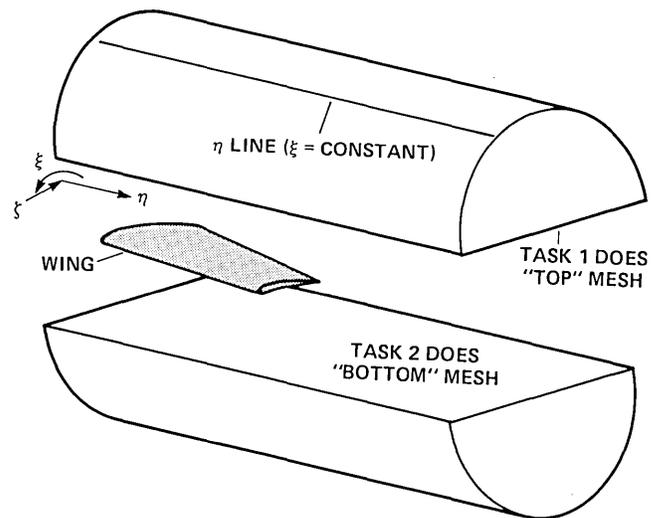


Figure 9. Computation divided in two different regions.

the three-step approximate-factorization scheme outlined above operating on successive "k-shells" in the mesh, beginning at the surface of the wing and progressing to the outer boundary. For each k-shell, the code:

- (1) fetches the appropriate subset of metrics from the disk
- (2) computes the density at each point
- (3) generates the special density coefficients
- (4) computes the residual terms resulting from equation 1.c
- (5) solves for $G(i,j)$ and $F(i,j,k)$

After completing this "k-loop," the code completes step three of the scheme by calculating the correction $C(i,j,k)$ and applies it to each mesh point to generate a new solution. A convergence check follows: when satisfactory convergence is achieved, the final solution is written to disk.

The Modification of TWING

TWING is written in portable FORTRAN 66 and executes on Cray, CDC 7600, and VAX CPUs. The program was rewritten to be well-structured. Its control flow is serial (i.e., few GO TOs jumping control around). Although it was possible to partition the computation along functional lines in a sort of high-level *pipeline*, this approach was not pursued because it needs either substantial additional memory or elaborate internal buffering to store intermediate results. Pipelining may also hinder efficient execution-time load-balancing with some stages of a pipeline executing longer than other stages of the pipe.

This problem was exacerbated in TWING by the extensive use of EQUIVALENCE statements in the original code, employed in an effort to squeeze the largest possible problems into the limited core memory of a CDC 7600 or a Cray 1S. Since a functional partitioning of the problem seemed unsuited to the limited shared memory available, a static spatial-partitioning scheme was employed.

Our restructuring took advantage of existing code and attempted as little algorithm change as possible. In this scheme, each step in the algorithm was examined in an effort to determine if several portions of the mesh could be operated on simultaneously at that step. Execution profiling using the Cray FLOW-

TRACE facilities showed dominant run times in three SUBROUTINES. Vectorized TWING executed three times faster than scalar TWING with input-output overhead included. Since distinct steps in the algorithm tend to correspond to separate modules in the finished code, this process resulted in a body of code that formed the skeleton of the concurrent processing portion of the modified TWING.

The calculations of the density (subroutine RO), the special density coefficients (subroutine ROCO), and the residuals (subroutine RESID) were all split along the η axis for each "K-shell" in the computational mesh (figures 6 and 7). This resulted in splitting loops (figure 10). One processor generated these results for points on or between the symmetry plane boundary and the wingtip. The other processor handled points on the wing extension, out to the freestream sidewall boundary. This "inboard-outboard" partitioning scheme was chosen because the algorithm employed in each of these calculations is usually constant for a given ξ line ($\eta =$ a constant) but varied with position along the η axis. An inboard-outboard scheme was therefore constructed using processor-dependent *branches* such as:

```

C
  IF (TASKID.EQ. 2) GOTO 12
  DO 10 I = 1,NIM
    . . .
  10 CONTINUE
C This continue added for multi-tasking
  12 CONTINUE

```

Mathematically, however, each point in the mesh was operated on independently during these preliminary calculations. We can replace the mesh with different divisions if there were reasons for favoring it.

A more fundamental relationship between the underlying mathematics of the algorithm and the spatial decomposition of the problem for Multiple-Instruction stream, Multiple-Data stream (MIMD) execution is illustrated by the three-step approximate-factorization scheme outlined in a previous section. Recall that in equation 2.a, the backward differencing is performed only about η , which generates tridiagonal linear systems along η lines ($\xi =$ a constant). This makes the inboard-outboard partitioning scheme used above unworkable for this step.

Table 1. Execution Time Profiling

Subroutine	Vectorized TWING [†] % Total Run Time	Scalar TWING % Total Run Time
RO	15.93	14.92
ROCO	13.45	15.91
RESID	23.92	17.99
Total %	53.3	48.82

†To clarify: this is not % of vector execution.

```

...
C Variables declared as integer, TASKID obtained from TSKVALUE.
  IF (TASKID.EQ. 1) THEN
    ROJSTART = 2
    ROJSTOP = NJTM
  ELSEIF (TASKID.EQ. 2) THEN
    ROJSTART = NJT
    ROJSTOP = NJM
  ENDIF
C The values of NJTM, NJT, and NJM are preset parameters
C in uni-tasked TWING.
...
C Now, each process works on the j-lines defined by
C the initial assignment block.
C
  DO 20 J=ROJSTART1,ROJSTOP1
C   DO 20 J=2,NJM -- old statement
    DO 15 I=1,NIM
...
15 CONTINUE
20 CONTINUE
...

```

Figure 10. Code illustrating the splitting of a loop.

However, adjacent η lines are computationally independent at this step, implying that the mesh could partition into "top" and "bottom" sections, each handled by a separate processor (figure 9). Similarly, in step two (equation 2.b), the backward differences are taken about ξ , generating tridiagonal linear systems along ξ lines ($\eta = \text{a constant}$) through the mesh. Here, each ξ line is computationally independent, and the resulting tridiagonal systems are solved concurrently by dividing the mesh into the inboard and outboard sections described in the last paragraph (see figure 6).

Finally, in step three (equation 2.c), bidiagonal systems are generated along lines in the ζ direction (ξ and η both = constants) (see figure 9). Again, concurrent processing of multiple ζ "pencils" is a simple and powerful way to use an MIMD machine at this step.

Note that true MIMD capacity was required to use such a spatial partitioning scheme. A vector architecture alone would not suffice because there was no guarantee that the instruction stream to be executed would be the same at different points in the mesh. Split difference schemes have sometimes proved useful. The wing root could

have used a more complex differencing scheme than employed near the outer boundary of a mesh. It is also possible that the values of some program parameters might also be position dependent.

Another code sequence commonly encountered in TWING was the selection of the maximum or minimum value in an array following an operation on the elements of the array. While this search has been conducted in a serial mode by the main program after the subprocesses return, this considerably degraded the resulting speedup. A better approach was to have each subprocess locate the maximum or minimum element in its portion of the data base, and pass the indices of this value back to the main program. The main program needed only to compare the two passed elements to obtain a maximum or minimum over the entire data base. An example of such a coding sequence is shown within the next code section (figure 11) where numbered variables are TASK-determined and nonnumbered variables are global shared variables.

VAX Modification

Our first MIMD testbed used two VAX 11/780 minicomputers linked to one MA780 multi-ported, shared memory unit. Because the operation of the processors was

asynchronous, each with its own copy of the operating system running on a local clock, the configuration was best described as a "loosely coupled" multiprocessor. Although each processor retained its large virtual address space as local memory, the shared memory in the MA780 was not virtually addressable. Each MA780 unit could accommodate up to two megabytes of physical memory. The unit employed for this study was equipped with 256 kilobytes of physical memory.

The operating system in use at the time of the study was VAX/VMS (Version 3.1). VAX/VMS provides three facilities for inter-process communication across the shared memory link: event flags, mailboxes, and global data sections.

Event flags are allocated in thirty-two bit clusters and are manipulated using a variety of system-supplied routines. A process could set or clear individual flags and could wait for the logical AND or OR of a multiple flag mask. One drawback of VMS-event flag services for MIMD programming was that the flag operations were *not* indivisible (atomic). This can cause difficulties when an MIMD program uses shared memory. It required protection from simultaneous access by more than one process, especially if the number of competing processes is great. In the present study this problem did not arise, both because, at

```
IF (ABS(RMAX1) .GE. ABS(RMAX2)) THEN
  IF (ABS(RMAX1) .GT. ABS(RMAX)) THEN
    RMAX = RMAX1
    IRMAX = IRMAX1
    JRMAX = JRMAX1
    KRMAX = KRMAX1
  END IF
ELSE IF (ABS(RMAX1) .LT. ABS(RMAX2)) THEN
  IF (ABS(RMAX2) .GT. ABS(RMAX)) THEN
    RMAX = RMAX2
    IRMAX = IRMAX2
    JRMAX = JRMAX2
    KRMAX = KRMAX2
  END IF
END IF
```

Figure 11. Selecting a maximum value from two locally determined maxima.

most, two processes were active simultaneously and also because they generally operated on different parts of the statically partitioned data base.

The VAX/VMS system was not intended to be a multiprocessor operating system. Programming the shared memory was clumsy. Since our shared memory was small, we reduced the resolution of the program to fit the space of the memory. This was a development measure that did not happen on our Cray. This paper does not cover the VAX specific version in any greater detail.

The other MIMD testbed consists of a Cray X-MP/22 running version 1.13 of the Cray Operating System (COS). The Cray, by way of contrast, is a "tightly coupled," shared memory multiprocessor. This creates problems not faced on our VAX testbed such as more memory contention but simplifies programming.

Cray Modifications

The VAX version of TWING was a "stripped-down" version of the production Cray code designed to fit into the small shared memory system. We, therefore, did not count on the VAX version to reach convergence. The mesh was too coarse, and we did not get a chance to truly debug the VAX version. The mathematical basis for partitioning the vector version of TWING (VTWING) was identical to the VAX-specific version. This time, we sought realistic convergence. Debugging was a major problem not only for TWING, but also for the new STACK allocation and multi-tasking of the CFT compiler we were testing.

One important side step, was a quick set of checks regarding the new SUBROUTINE linkages. We should mention this was not a problem for TWING. To do this, a user compiled the complete, existing program using the ALLOC=STACK option on the new CFT compiler. The program was then run using the associated new loader given adequate stack and heap sizes (see the manual) (Research, 1985). The results were compared to the original STATICally compiled run. A useful variation of this was to create simple TASKs that START then immediately WAIT as a CALL to a SUBROUTINE would:

change from: to:
CALL RO CALL TSKSTART(TA.RO)
CALL TSKWAIT(TA)

The timing differences between STATIC and STACK runs are included in the section on **Performance**. The compiler changes affect program execution without source code changes.

The next stage entailed converting the existing code into a multi-tasking body of code. This was not as easy as it appeared as subtle errors required detection and correction. It is possible to do this at different levels or stages such as converting the entire program, converting subroutines, or converting blocks of code. Converting a code in large sections is like writing a large program and expecting it to run correctly the first time.

It was important to have good comparison data, since fast execution did not imply correct execution. A machine-readable output was created from an unmodified, running version of TWING. Once the code was running, we tested the output of the multi-task run with our uni-task output using a differential file comparator (the UNIX⁷ *diff* program). This insured that our conversion was precise.

Our third and last attempt at conversion was to break a subroutine into two smaller subroutines: a parallel portion and a serial portion. Since most of the data was stored in COMMON blocks, parameter passing was minimized to simplify these problems. The parallel subroutines were run and synchronized before the serial portion as shown in figure 12.

Portions of serial subroutine code (typically loops) then migrated to the parallel subroutines. This technique successfully identified subscripting oversights, branching problems, and so on. It was painfully slow, but it was effective. Initially, task synchronization was performed using TSKSTART and TSKWAIT, not the more complex EVENT flags. We used the "Make it right before you make it faster" philosophy from the *The Elements of Programming Style* (Kernighan, 1978).

We stress the following point: make certain that the existing code is bug-free. There

⁷UNIX is a trademark of AT&T Bell Laboratories.

```

...
MESH = 1
CALL SCHED
...
SUBROUTINE SCHED SUBROUTINE PROCES
...
CALL EVPOST(GO) CALL EVWAIT(GO)
... IF(MESH.EQ.1) THEN
CALL EVWAIT(DONE) CALL RO
...
... CALL EVPOST(DONE)
END
...
END

```

Figure 13. Structure of our simple scheduler.

loops split across processors compared to TWING. We split a total of 19 loops in three SUBROUTINES. This includes new code for loop splitting, new per-process branches, TASK-EVENT creation and control code, and a small TASK scheduler. About 210 lines of control flow code were added (excluding comments). 70 more lines were replaced or modified into 160 lines to handle problems of parameter passing, or changes to array indices.

During the development of each TASK, good version control proved useful. A good tool requires parallel branching versions; linear version control such as UPDATE was not adequate. Maintaining the successful, intermediate stages of multi-task TWING made debugging and scale-up easier through the isolation of changes. It was always possible to easily fall back to some parallel, executable code.

Debugging

Sequential debugging is generally regarded as a *black art*. Bugs occur during compile-time and run-time: with the latter, the non-fatal ones are the hardest to find. The basic techniques for debugging are categorized into: 1) traces, 2) snapshots or 3) dumps. These techniques have problems in multiprocess environments lacking consistency or having deadlock. Multi-task debugging is plagued by a lack of reproducibility, synchronization, and good tools. The literature on run-time debugging in multiprocess environments is scarce (Model, 1979) and more work

is needed in this area.

Numerous users tell us to “force multi-task execution into a single stream of execution⁹” as if simple user-controlled reduction would solve hazard problems.

This does not help!

Normal debugging depends on a machine being in a reasonably consistent state. A multi-task program crash may not occur at the same location as with a uni-task program. This is true for uniprocessors executing multi-task code as well.

Consider a simple example to illustrate the conceptual difficulties of debugging using the CFT traceback facility. A program creates a *child* TASK. When the child TASK dies, should the traceback trace through the point where the child process began, or should it trace through the synchronization routines (if any)? The tangled nondeterministic web makes this decision difficult. There are situations where one trace is preferable over the other. One condition is when the child dies because of the actions of its *parent* or *sibling* processes [side effects]. So, traces are not simple. What about snapshots?

⁹This is accomplished using the TSKTUNE call and setting the MAXCPU parameter to '1.'

Inserting WRITE statements into programs might not help. First, the execution order of these statements may vary (e.g., non-determinism). Second, I/O is another shared resource, and the user must have LOCKs that protect that resource like any other shared resource.

One surprising effect of inserting WRITE statements at key points was the migration of bugs from one location to another! We solved this debugging problem by modifying our technique of migrating code between serial and parallel development sub-routines. Our new technique was to remove data structures and code immediately following the breakage point to isolate program and compiler bugs. This sometimes worked to locate bugs. The problem at this point becomes: is the program crashing because of the original bug or the bugs introduced by cutting code?

In the I/O locking process, it would help users debug codes if the system could hide I/O locking details from users. Better yet, a small library of simple routines would help. It should have traceable ERROR and ASSERTION routines. If a user resorts to adding WRITE statements to follow the execution of a program, the user should have a similar trace of a serial code for advanced comparison. A simple filter could take a source program and insert a WRITE with the subprogram's name. More elaborate and more powerful debugging tools would also help.

Dumps, the method of last resort, are frequently less consistent than traces or breakpoints. We avoided dumps at all cost.

One technique tried in the latter stages of multi-task conversion was program proving. Toward completion of program scale-up, we had a tricky change to a SUBROUTINE call. Precondition and postcondition assertions were compiled surrounding critical code changes. Proof techniques had limitations in a parallel environment, but it was a useful technique for checking changes. Program proving was not regarded as a cure-all and was regarded as controversial.

The last set of problems involves synchronization and timing. A new diagnostic message for first-time multi-tasking programmers is compressed from a real CRAY job in figure 14. *Race conditions* occur whenever two or more TASKs or processors are sharing

data (or code). This is the time when *deadlock* can occur. There are no general solutions, but there is a mountain of research literature. Multi-tasking CFT provides limited deadlock detection and traceback. Keeping TASK scheduling and timing constraints simple is currently the best way to avoid deadlock. The most difficult deadlock problems should occur when there are indirect deadlocks.

Testing Multiprocessor Outputs

A running multi-task program was not enough; we sought numerical results identical to our uni-task TWING. There were many occasions where our program ran to completion, but our numbers did not agree at lesser digits of precision. A standard file comparator was used to test output between TWING runs. The importance of tools such as a good file comparator was not underestimated. A single, incorrect, boundary subscript could "poison" an entire array. Testing asynchronous methods [e.g., chaotic relaxation] is more difficult.

Fortunately, our program is completely synchronous. However, newer asynchronous, chaotic algorithms remove the consistency assumption and approximate a solution. If such asynchronous methods are used, file comparator programs are completely inadequate. Better comparison tools are needed. Output testing tools must approximate floating-point comparisons within a specified tolerance.

The Cray multi-task version of TWING had proved our concept by reaching convergence with results identical to a uni-task version of vectorized TWING.

Other Generally Useful Tools

While mentioning debugging tools, we should also mention other generally useful tools. Among these we could include tools to search for STATIC allocation and data dependence. Data dependence tools can also provide help when recursion is added to FORTRAN. A good cross-referencing tool could aid this search process. Other tools could possibly identify linkage problems. Such tools are useful in the analysis and compilation phases of development. All these programs should execute independently (i.e., from a compiler) in the style of other good software tools.

```

USER UT024 - DEADLOCK - ALL USER TASKS WAITING FOR LOCKS OR EVENTS
USER TB001 - BEGINNING OF TRACEBACK
USER   - $TRBK WAS CALLED BY UTERP% AT 1715731a
USER   - UTERP% WAS CALLED BY $SUSTSK% AT 1705627a
USER   - $SUSTSK% WAS CALLED BY EVWAIT AT 1701511b
. . .
USER TB002 - END OF TRACEBACK

```

Figure 14. A frequent error message for new users of multitasking.

Performance and Execution Behavior

The measurement of parallel programs is conceptually complicated by several factors. The Cray measurement facilities, if used, record the length of all parallel execution traces as if they were measured sequentially. For instance, two cycles run in parallel take one cycle to execute, but they are still counted as two cycles. The Cray documentation (Research, 1985) notes that flow tracing facilities do not work properly with multi-tasking environments. We resort to the direct use of the system real-time clock and flow tracing of the uni-task version of TWING to give us run-time characteristics.

There are no standard metrics for determining multiprocessor performance improvement. The most common in use is simple *speed up* defined by:

$$\text{Simple speed-up} = \frac{\text{Serial execution time}}{\text{Parallel execution time}}$$

The simple speed up of TWING is illustrated in the next table.

Another conceptual measurement problem is where and how measurements are taken. We simply throw two CPUs at a problem, so the maximum simple speed-up is one-half the total serial execution time. I/O wait time is a significant portion of the program that cannot multi-task. We recognize that we don't use two CPUs for the entire time: we have serial code, and we have wait-time for TASKs to finish and synchronize. Also, we need more cycles to cover overhead.

Since we were able to multi-task only 50% of the total serial execution, the best improvement we could gain would be 25% of total execution. We might term this

performance figure as *proportional*, simple, speed-up. As we multi-task more code, this figure should slowly increase.

Still another problem is that with two or more CPUs sharing common resources - memory and I/O - collisions become inevitable. Processors are forced to wait, and this expends more overhead cycles. This contention is visible when running a uni-task version of the code in one processor, and running a second code in another processor. By varying the work load in the second program between a CPU intensive versus memory intensive JOB, we can see the simple, but significant effects of memory contention (See the Table below). These are interference effects not found on uniprocessors. A problem arises in shared memory multiprocessors such as on our VAX and Cray that local memory multiprocessors do not have. Memory contention significantly slows down memory performance. Designers of future multiprocessors must balance processor- versus memory-performance rates.

Another performance issue is the addition of overhead cycles required to control TASKs. Figure 15 shows the cost in cycles versus the iterations toward solution for our VAX version. This cost occurs similarly on the Cray.

Load balancing is a significant problem since TASKs vary in work load, and we have seen that measurement of load has problems. The output from the Cray day files shows a considerable imbalance of work. Cray tools discovered that TASK 1 did more work (executed significantly longer) than TASK 2. The timing output of a single day-file illustrates the difference on our two CPU system:

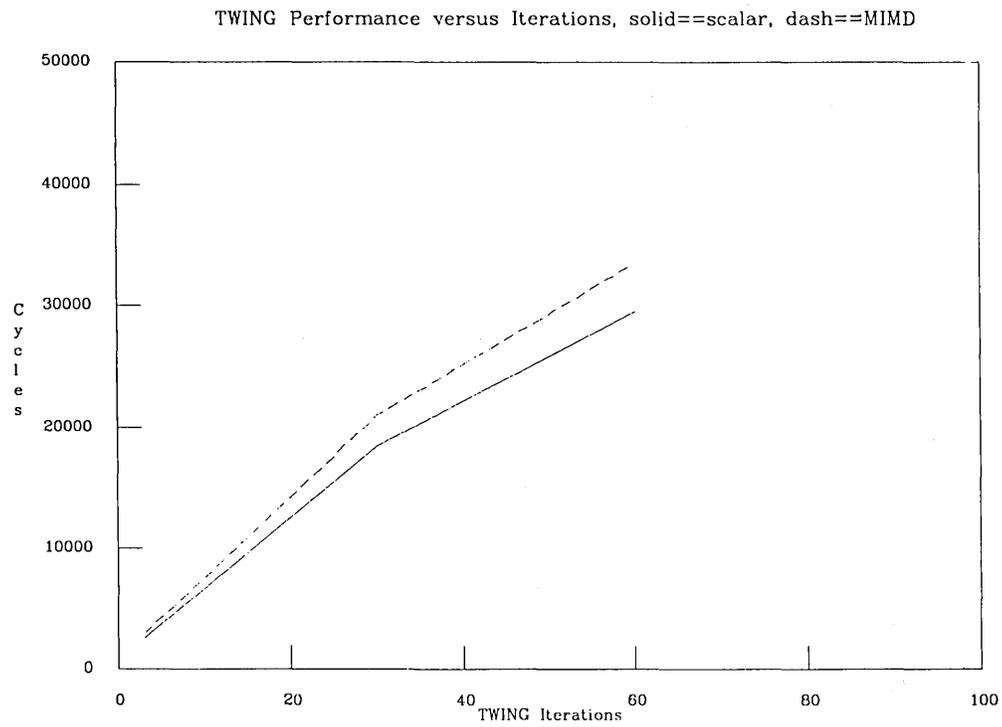


Figure 15. Graph showing the additional overhead (near linear) between sequential versus parallel code versions.

CPU's used	STATIC Compile one	STACK Compile one	Multi-tasked twc	Speedup **
Real wall time	7.76	7.17	7.16	10%
Total system time	7.36	6.76	9.67	n/a†
Input	0.0244	0.0255	0.0263	n/a†
Init	0.250	0.210	0.211	n/a†
Solve	7.08	6.52	9.43	n/a†
RO	1.25	1.06	1.23	2%
ROCO	0.985	0.891	0.886	10%
RESID	1.75	1.73	1.42	20%
†not applicable: sequential FLODUMP timings added only for completeness: the difference in wall clock time versus what the operating system reports.				

	Low Memory Contention		High Memory Contention	
	STATIC compile	STACK compile	STATIC compile	STACK compile
Real Wall Time	7.94	7.36	8.49	8.04
Total-System Time	7.35	6.76	8.03	7.35
Subroutines:				
Input†	0.0244	0.0255	0.0264	0.0276
Init†	0.250	0.210	0.270	0.225
Solve	7.08	6.52	7.73	7.09
RO	1.26	1.06	1.35	1.13
ROCO	0.995	0.893	1.07	0.970
RESID	1.77	1.74	1.91	1.89
†These SUBROUTINES were not converted to use multi-tasking. They are included here for control reasons to show the effect of changing to a STACK compilation.				

TASK	CP TIME
1	11.85
2	4.83

This is because the work areas were not partitioned evenly between the two TASKs based on hand analysis of array proportions. Work was partitioned based on existing, somewhat lopsided DO-loop parameters in three-

dimensional arrays.

To change these parameters would require more computation and potentially further array-subscript change. Additional algorithm modifications are required for boundary regions. Dynamic load balancing is harder still.

Discussion

Further Research

This research has not covered other forms of multiprocess partitioning. Pipelines are a common proposal: easily constructed and debugged, but difficult to tune or load balance. (See **Scale-Up**.) The program's author (Thomas) is considering this approach, but it requires extensive rewriting.

Micro-tasking is another Cray-proposed multiprocessing construct (Booth, 1985). Micro-tasking involves a simpler, more restrictive set of control primitives. Another important issue is the area of scale-up (See next section).

Scale-up

Certain aspects of scaling up programs are trivial. Increasing problem size is not typically a problem: our VAX case was not a necessary prerequisite to move the program to the Cray. Adding more processors, however, is not trivial. The work on the TWING code began before there was any consideration of generalizing the program to use more than two processors.

The current multi-task work on TWING will not generalize to an n -processor case. The code used to determine maxima is one problem that will not easily scale. If more than two processors are used, different partitioning schemes become preferable.

Probably the key issue of multi-tasking is whether the performance gained was worth the effort expended. There is a conflict (or tradeoff) between the need to have large multi-task sections for performance and small multi-task sections for ease of development and debugging.

The multi-tasking programmer must also confront the need to have large protected critical sections and many asynchronous processes running. Our scale-up of the code uncovered many machine-dependent assumption problems. For the scale-up of code, the parallel-serial divide-and-conquer approach again worked.

Open Issues

The problems of automatic partitioning are not addressed in this study. Our future intent is to extend FORTRAN by using a

simple preprocessor to add support for simpler constructs (e.g., COBEGIN, COEND) like Cray micro-tasking. The preprocessor should ideally hide low-level details and machine dependent processing. It is tempting for programmers to be parochial about particular constructs, so we wish to avoid this by using preprocessors. Similar research is under study on different architectures at other sites (e.g., LANL, ANL, Bell Labs, CMU, U. of Ill.).

There are dozens of issues left open: different synchronous and asynchronous algorithms, translation into an intermediate language for dataflow-style execution, measurement and load balancing. Parallel processing has many difficult problems remaining which will take years to research.

Conclusions

The introduction of parallelism is as significant a tool as either Cray multi-tasking or micro-tasking. The problems of parallelism are not new. They are typically thought to inhabit that realm called *systems programming*. Users intending to add parallelism to their collection of tools are advised to learn from experience of others.

Good software tools would help programmers. These tools must provide multiprocessing support. Many programmers would probably desire a standardization of multiprocessing syntax, but this is premature.

Programmers should recognize that with adding parallelism and achieving better performance, there will come some loss of the coherent sequence that makes sequential programming such a powerful tool.

Programs designed to use parallelism from their inception are more likely to use parallelism efficiently. This was clearly the case with the introduction of vectorization, i.e., vector-designed programs tend to use vectors more efficiently. We should soon see more multi-task programs, but it is an open question whether these programs are scale-able into the hundred- and thousand- (proposed) processors range.

Acknowledgements

Michael Johnson deserves special recognition for doing the preliminary VAX work (Johnson, 1983). Scott Thomas and Alan Fernquist provided valuable assistance with

uni-task TWING. Ken Stevens, Eric Barszcz, and Cathy Schulbach assisted with debugging. Dave Robertson of Zero-One relayed our many CFT problems back to Cray Research.

References

- [Larson, 1984]
John L. Larson. "Multitasking on the CRAY X-MP-2 Multiprocessor." *Computer* **17**(7), pp. 62-69 IEEE. (July 1984).
- [Research, 1985]
Cray Research, Inc., "Multitasking User's Guide," Technical Note SN-0222, Rev. A (January 1985).
- [Research, 1984]
Cray Research, Inc., "Multitasking and the X-MP." Technical Note (January 1984).
- [Dennis, 1979]
Jack B. Dennis. "The Varieties of Data-Flow Machines," *1st International Conference on Distributed Computing Systems*, pp. 430-439 IEEE. (October 1979).
- [Denning, 1985]
Peter J Denning, "The Science of Computing: Parallel Computing," *American Scientist* **73**(4), pp. 322-323 (July-August 1985).
- [Jones, 1980]
Anita K. Jones and Edward F. Gehringer, eds., "The Cm* Multiprocessor Project: A Research Review," CMU-CS-80-131, Carnegie-Mellon University, Pittsburgh, PA (July 1980).
- [Grit, 1983]
Dale H. Grit and James R. McGraw. "Programming Divide and Conquer on a Multiprocessor." UCRL-88710, Lawrence Livermore National Laboratory, Livermore, CA (May 1983).
- [Jones, 1980]
Anita K. Jones and Peter Schwarz, "Experience using multiprocessor systems - a status report," *Computing Surveys* **12**(2), pp. 121-165 (June 1980).
- [Kuck, 1980]
David J. Kuck, Robert H. Kuhn, Bruce Leasure, and Michael Wolfe. "The Structure of an Advanced Vectorizer for Pipelined Processors," *Computer Software and Applications Conference (COMP-SAC80)*, pp. 709-715 IEEE, (October 1980).
- [Andrews, 1983]
Gregory R. Andrews and Fred B. Schneider. "Concepts and Notations for Concurrent Programming," *Computing Surveys* **15**(1), pp. 3-43 (March 1983).
- [Dijkstra, 1968]
E. Dijkstra. Multiprogramming System"" "The Structure of the "THE" Multiprogramming System." *Communications of the ACM* **11**(5), pp. 341-346 (May 1968).
- [Hoare, 1974]
C. A. R. Hoare. "Monitors: An Operating System Structuring Concept." *Communications of the ACM* **17**(10), pp. 549-557 (October 1974).
- [DOD, 1980]
DOD, *Reference Manual for the Ada Programming Language*, U.S. Department of Defense (July 1980).
- [Dijkstra, 1968]
E. Dijkstra. "Go To Considered Harmful," *Communications of the ACM* **11**(3), pp. 147-148 (March 1968).
- [Fong, 1984]
Kirby Fong, *Personal Communication*, LLNL, Magnetic Fusion Energy Computer Center (1984).
- [Thomas, 1983]
Scott D. Thomas and Terry L. Holst, "Numerical Computation of Transonic Flow About Wing-Fuselage Configurations on a Vector Computer." *AIAA 21st Aerospace Sciences Meeting*, (January 1983).
- [Kernighan, 1978]
Brian W. Kernighan and P. J. Plauger, *The Elements of Programming Style, 2nd edition*, McGraw-Hill, New York, NY (1978).
- [Model, 1979]
Mitchell L. Model. "Monitoring System Behavior in A Complex Computational Environment." CSL-79-1. Xerox Palo Alto Research Center, Palo Alto, CA 94306 (January 1979).
- [Booth, 1985]
Mike Booth. "Microtasking Presentation," Internal Report, Cray Research

Inc.,
Dallas, TX (1985).

[Johnson, 1983

Michael S. Johnson. "Modification of the
TWING Full Potential Code for Execution
on an MIMD Computer." Tech.
Memo., NASA Ames Research Center,
Moffett Field, CA (1983).

**SPEEDUP PREDICTIONS FOR LARGE SCIENTIFIC PARALLEL PROGRAMS
ON CRAY X-MP-LIKE ARCHITECTURES**

Elizabeth Williams
Frank Bobrowicz

Los Alamos National Laboratory
Los Alamos, NM

ABSTRACT

How much speedup can we expect for large scientific parallel programs running on supercomputers? For insight into this problem we extend the parallel processing environment currently existing on the Cray X-MP (a shared memory multiprocessor with at most four processors) to a simulated N -processor environment, where $N \geq 1$. Several large scientific parallel programs from Los Alamos National Laboratory were run in this simulated environment, and speedups were predicted. A speedup of 14.4 on 16 processors was measured for one of the three most used codes at the Laboratory.

LOS ALAMOS NATIONAL LABORATORY CONTROL LIBRARY

F. W. Bobrowicz

Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

The Los Alamos National Laboratory multitasking control library is now being used in the development of multitasked computer programs. Extensions, modifications, and improvements that have been made to this library as a result of these experiences will be discussed. Results obtained for some of our multitasked codes will be presented.

NETWORKING AND FRONTENDS SESSION II

Dean W. Smith

ARCO Oil and Gas Company
Plano, Texas

This session consisted entirely of a number of talks by Cray Research personnel responsible for the development and support of Cray's station products. The following presentations were made: MVS station status by Peter Griffiths, Superlink status by Stuart Ross, APOLLO and VAX station status by Martin Cutts, VM and UNIX station status by Allen Machinski, and Cyber station status by Wolfgang Kroj. The following sections are based on by notes and the overheads from the individual's presentation.

MVS STATION STATUS
Peter Griffiths
Cray UK

The current MVS station release is 1.13 bugfix 2 which was released in October of 1985. A major new release is scheduled for the first quarter of 1986. The numbering scheme for the MVS station will be changed at that time and the new release will be known as 2.01. Minor releases are scheduled for 2Q86 and 4Q86, and a major release is also scheduled for 3Q86.

Release 2.01 of the MVS Station has a number of new features addressing interactive support, multi-cpu support, RACF support, operations support, installation exit invocation improvements and station tape processing.

The multi-cpu support will address the problem of operating a single station in a complex that has more than 1 processor. Currently, the station provides little support for users who may not be executing on the same cpu as the MVS station. The new station will support data and communications paths in a loosely coupled cpu configuration (the shared database/shared spool environment). It will also support a communications path in a remotely configured cpu environment (ACF/VTAM network). This feature should greatly enhance the exposure of the CRAY and extend the CRAY environment throughout a large MVS network.

A closely related feature is the SCP interactive support. Long a feature of the SCP protocol, the MVS station will now support interactive sessions on the CRAY from an ACF/VTAM network.

RACF support will be enhanced by two features. RACF 1.6 will be supported which, when installed with the TCB extension feature, removes the need for the station to perform its own RACHECKs and RACDEFs. Additionally, RACF support will be extended to the protection of COS tapes.

Several operational enhancements have been added to the new release. Jobs can be held prior to transfer, there will be an automatic logon and relogon feature, MVS I/O will use QSAM, and there will be pre-mount messages for station tapes.

Installation enhancements include: the installation exit invocation improvements, and the JES2 modifications will be packaged as JES2 exits.

This new release contains many features which have been desired in the MVS environment for a long time.

SUPERLINK STATUS
Stuart Ross
Cray Research UK

The superlink project actually consists of a number of products. The current product is Superlink/ISP. Its follow-up product, seen as the marriage of the Superlink/ISP and the MVS station product, is known as Superlink/MVS.

Superlink/ISP was supported in COS 1.14 BF2 and is currently in Beta test status (see discussion by Ronald Kerry on ISP in the Networking Session proceedings).

Currently the product is undergoing product stabilization, benchmarking and analysis of customer experiences.

Superlink/ISP's features include a high performance data pipe, sequential data access to MVS datasets, and access to any peripheral on an IBM system.

Superlink/MVS is expected to provide a number of new features in the areas of direct data access, communications access, network interfaces and application capabilities. The concept is to provide a layered architecture that can satisfy a wide range of requirements and applications.

The new network interface will utilize a network access method based on an ISO model for open systems interconnection; support a number of interface devices; provide a high performance transport service in the operating system; and establish a transportable applications interface.

In the realm of data access, direct record-level I/O, both sequential and random, will be provided as part of the standard product. Superlink/MVS will serve as the communications access interface by being a gateway to VTAM and supporting CRAY interactive sessions.

Because Superlink/MVS will provide an applications layer, the distinction between data and control information should blur, allowing users to establish their own process to process communications. Eventually, it should be possible to develop distributed applications where work is being performed on the host system and the CRAY in a coordinated method on whichever system performs that function best.

APOLLO AND VAX STATION STATUS
Martin Cutts
Cray Research UK

The current release of the APOLLO station at the time of the Montreal CUG was Release 1.01, which was released in May 1985. Release 1.01 supported job submission, dataset staging of character blocked and binary blocked (CB and BB) data formats, and SCP interactive sessions.

Release 1.01 of the APOLLO station is scheduled for November 1985 and is planned to support improved staging performance, direct dispose of graphics, and support for AEGIS release 9.0.

Release 1.03 is to be a bugfix release and no features are expected to be provided.

Release 2.0 of the APOLLO station is scheduled for the second quarter of 1985 and is expected to complete the user

interface. Features expected are: menu driven user interface; operator control for STATUS, DROP, JOB, and KILL; as well as support for transparent dataset staging.

The current VAX/VMS station release is 3.01 and was available in October 1985. Features included in the new release are:

Excluding interactive identical user environment whether attached or remote.

Multiple stations on one VAX.

Ease of installation using command procedures and configuration utility.

Control and authorization managed from attached station.

Efficient network dataset transfers.

Spooling.

Range of commands extended.

Release 4.0 is scheduled for fourth quarter of 1986 and is expected to support features in two different VAX environments.

Clustered Stations:

Decnet not used within a cluster.

Cluster wide queues.

Only one copy of the station with a cluster.

MICROVAX/VAX Station 2 Support:

Remote interactive.

Menu driven user interface.

Separate interactive graphics window.

VM and UNIX* STATIONS
Allen Machinski
Cray Research

Allen Machinski addressed the station products in the IBM VM and VAX UNIX stations.

The current VM station is release 3.0 and was available in November 1985. New features available in the release were:

Tape enhancements: label processing, user exits and multi volume/multi dataset support.

Intertask dispose.

Graphics support, both interactive and by dispose.

Dispose to CMS files (mini disks).

Performance enhancements including performance statistics reporter and real time monitoring and diagnostic aids.

Improved documentation with online error messages and online manuals using DCF.

MVS NETDATA job/dataset submission and MVS TSO/E support.

The next release of the VM station (R 4.0) is expected to provide the following features:

VM/SP release 4.0 support.

VM/SNA considerations.

Application program interface for dataset staging and station messages.

Device statistics monitor.

Dataset transfer accounting.

Variable segment size.

CRSTAT fullscreen reads.

Disk acquire/fetch performance enhancement.

Allen's second product discussion reviewed the status and features of the UNIX station in the VAX environment.

The UNIX station was written in the C language for transportability. It supports both batch and interactive facilities. It has a software requirement of ATT UNIX 5.2, and a hardware requirement of the NSC HYPERchannel interface. The UNIX station consists of two components, the station concentrator and the user interfaces to the station concentrator.

Concentrator Features:

One concentrator per destination CRAY.

Separate batch and interactive processes.

File transfer features: multiple streams, job submission from frontend, dataset staging to and from CRAY, and multifile dataset support.

File transfers survive station or COS failures.

Maintains log file for batch and interactive processing: date and time of logon, stream activity, error conditions and debug messages.

Optional trace file of messages for debugging: enable or disable while station running; trace any or all message codes; and trace NSC header, LCP and segments.

User Interface features:

Separate input and output processes.

User can set default destination CRAY.

Escape to shell.

Maintains logfile for each user.

Interactive command syntax very similar to operator console.

Input and output redirection.

Local commands: CRAY, HELP, END, REFRESH and SCROLL.

Display commands: DATASET, JSTAT, STATCLASS, STATUS, and STORAGE.

Dataset commands: SAVE, and SUBMIT.

Operator commands: CHANNEL, CLASS, DEVICE, DROP, ENTER, FLUSH, KILL, LIMIT, OPERATOR, RECOVER, RERUN, RESUME, ROUTE, SHUTDOWN, STREAM, SUSPEND, SWITCH.

* UNIX is a trademark of AT&T Bell Laboratories

CRAY CYBER LINK SOFTWARE
PRODUCT STATUS AND DEVELOPMENT PLAN
Wolfgang G. Kroj
Cray Research GmbH

The major points of Wolfgang Kroj's talk included a discussion of the NOS station, NOS/BE station, NOS station interactive support, Control Data Corporation hardware and software environments, dual state support, station support policy and a statement of direction.

The current NOS station release is 1.14 and became available in May 1985. Features included in that release included:

NOS 2.2 and NOS 2.3 support.

U of M PASCAL release 4.0 support.

Multiple station support.

Runtime driver selection.

History trace facility.

The next release of NOS station 1.15 will become available in the 4th quarter 1985 and its new features will include:

Enhanced history trace facility.

Operator command for stream control.

Dual state support.

NOS station release 1.16 is expected in 2nd quarter 1986 and the only expected feature is interactive support.

provides full implementation of the SCP interactive frontend protocol.

Design requirements: optional feature, restricted access, preserve overall station performance and reliability, minimal changes to existing station.

Functionality similar to VM and VMS stations.

Multi CRAY support.

The current NOS/BE station release is 1.14 and became available in September 1985. Features included in that release included:

History trace facility.

Runtime driver selection.

Operator command for stream control.

U of M PASCAL release 4.0 support.

The next release of NOS/BE station 1.15 will become available in the 4th quarter 1985 or 1st quarter 1986. Its new features will include:

Enhanced history trace facility.

Enter command.

Dual state support.

The CYBER 180 represents a departure for Control Data and Wolfgang reviewed some of its hardware and software features. The CYBER 180 architecture has 64 bit words, 8 bit ASCII character set, hexadecimal internal representation, two's complement arithmetic, IBM-like instruction set, microcoded machine, cache memory, byte-addressable virtual memory, 4096 segments of maximum 2GB, and "wired MULTICS". The NOS/VE operating system

includes the following features: supports new 180-type hardware, dual state, multi-tasking, interactive, tools, written in CYBIL, binary releases, and CDNA/CDCNET.

The features of dual state support include:

Provides basic CRAY station functionality to dual state users.

Release 1.15 of the Cyber stations.

Installation option.

Features provided for NOS/VE users: CRAY job submission, CRAY system status, and coded file staging between COS and NOS/VE.

Requires minimum NOS/VE level of 1.1.2 for NOS dual state.

Station Support Policy:

The Cyber stations are intended to support future NOS-, and NOS/VE systems no later than 6 months after the respective release from Control Data.

At time of availability a particular Cyber station software release is intended to provide support for the last 2 levels of corresponding CDC operating system releases.

The previous release of the Cyber station software will be supported for at least 4 months following new release availability.

Bugfix releases will supercede all previous bugfix and critical releases for a specified release level.

The Cyber stations will continue to support the latest U of M PASCAL compiler release exclusively, subject to availability of at least 6 months prior to the station release.

Statement of Direction:

It is not planned to add new features to the NOS- and NOS/BE station software after release 1.16 and 1.15 respectively.

The station software supplied by Cray Research will support NOS- and NOS/BE dual state operation. Hooks for NOS/VE users to access basic station functions will be provided.

Cray Research plans to provide CRAY Cyber Link Software to support Control Data frontends running the NOS/VE operating system.

COS Session

David Lexton

University of London Computer Centre
England

A. COS Experience Panel

(i) Claus Hilberg (ECMWF)

(a) COS itself

The current system at ECMWF is COS 1.13 BF2 with a number of additional bug fixes, all of which have now been released through the SPR mechanism. There have been no software crashes for many weeks, and we consider today's COS 1.13 a very good system.

We plan to replace the current X-MP/22 with an X-MP/48 at the end of this year, so we are right now upgrading to COS 1.14. Because of our obligation to produce a 10 day global weather forecast every day, we test new software very thoroughly. We're pleased to say that at this moment there is only one outstanding COS problem, which we believe has to do with overflow of SSD files to disk, and which we also believe CRI will send us a fix for shortly. We have had little trouble fitting local code - the only major local mod is a simplified version of the NASA/Ames code to allow operators to initiate rollout of SSD space.

All in all we are very optimistic about COS 1.14 - it looks like a good system.

(b) Compilers

Much was already said at this CUG about errors in the CFT 1.14 compiler. ECMWF can add to this that CFT 1.13 was no better. We have in essence 3 different Fortran programs that are vital for our production, and we have not yet seen one compiler - be it 1.13 or 1.14 - that will correctly compile all 3 programs. We are very dissatisfied with CRI's performance in this area.

(c) Libraries

ECMWF uses multitasking for the operational forecast, and at the same time we run both STATIC and STACK calling sequence programs. This makes the maintenance of libraries a complex matter. Currently we try to maintain 5 different combinations of compilers and libraries:

- I for old calling sequence programs, we still run CFT 1.11 with libraries also at level 1.11.
- II for normal production, STATIC programs use CFT 1.13 BF2 with 1.13 BF2 libraries.
- III for normal production, STACK programs use CFT 1.14 BF1 with selected bug fixes, a 1.14 BF1 \$FTLIB, and other libraries at 1.13 BF2.
- IV we are testing 1.14 BF2 compiler and libraries for STATIC programs. This testing has been going on for some time, and we're still finding catastrophic errors.
- V we are also testing 1.14 BF2 compiler and libraries for STACK programs. They appear to work all right, but we cannot put them into production because they don't function easily without the new LOADER.

Having to look after so many compiler/library combinations requires much effort, in particular when users by accident pick up non-matching versions and then come to user support and ask for help.

(d) Compatibility

We feel that CRI is not paying sufficient attention to the problems of compatibility between releases. We mention two examples: First, CFT 1.14 BF2 will not function properly in a COS 1.13 environment because the runtime libraries use TASK COMMON, which is not understood by 1.13 LOADER, and produce some spectacular runtime errors; it didn't really have to be that way, one could with limited effort have produced library routines that avoided TASK COMMON in a 1.13 environment. And second, the 1.14 LOADER that would overcome the first problem won't work under 1.13 because it issues a new kind of F\$BEGIN calls; again, one could have made LOADER realise it was talking to a 1.13 environment and used different F\$BEGIN calls.

It should be a requirement that level N of the products must function with the operating system at level N-1 and level N+1. Fortunately, we have seen no problems running 1.13 products under COS 1.14. We would have been happier if this was by design, but we suspect that it is by good fortune only.

(ii) Conrad Kimball (Boeing)

(a) Software Configuration

Boeing Computer Services (BCS), a division of The Boeing Company, is currently running COS 1.12 on a Cray-1S/2000 and an X-MP/24. We provide the 1.12, 1.13, and 1.14 product sets to our users, and have 15,000 lines of local code in them. We have 60,000 lines of code in COS proper (much of it in two new tasks), and 180,000 lines of code in new system utilities. On our X-MP we are running the 1.13 level of the IOS software, with the User Channel Shell feature retrofitted from 1.14. We have 10,000 lines of code in the IOS, primarily in the form of an NSC A130 driver for our HYPERchannel local network.

(b) X-MP Installation

In the summer of 1984 BCS began planning for the installation of its X-MP. At that time we had a pair of 1/S's, and we were about to convert to COS 1.12. A major problem that we faced was the issue of binary compatibility - the instruction sets of the 1/S and X-MP are slightly different. In particular, CFT 1.09 and some of the \$SCILIB routines used the unadvertised vector recursion facility of the 1/S.

Since we were about to convert to COS 1.12, we decided to make the 1.12 libraries 'neutral' - that is, able to run the same binaries on either a 1/S or an X-MP. When our users converted to CFT 1.11 and the 1.12 libraries they would automatically get 1/S / X-MP compatibility. This neutrality was achieved by modifying a handful of \$SCILIB routines to make a runtime test to determine on which model they were running, and adjust their algorithms accordingly.

Another major problem was the IOS on the X-MP - our very first IOS. BCS has a locally developed HYPERchannel network, and we had to somehow move our NSC A130 driver out to the IOS. In addition, we had modified our Data General station to provide local commands and displays, and we were faced with how to provide them on an IOS.

To handle our local network requirements, it was decided that we should install a 1.13 IOS and retrofit a pre-release of the User Channel Shell driver from the 1.14 IOS. All we had to do then was provide an NSC A130 driver overlay in the IOS, interfaced with the User Channel Shell facility, and some EXEC modifications to interface with the COS side of the facility. Much to our relief we encountered only a few minor incompatibilities between COS 1.12 and the 1.13 IOS, which were quickly fixed. The User Channel Shell facility went in very easily, and has run for seven months with few problems.

Rather than modify the IOS operator station to provide displays and commands like those we had put into the Data General station, we chose to make use of the IOP interactive station facility. We built a utility program (and several new EXP functions) which runs at an IOP interactive console, and which provides our local displays and commands. The IOP

interactive facility has also tremendously improved the productivity of our test sessions.

Finally, the X-MP was to be delivered just four weeks before the scheduled production date - not much time for software development and checkout, considering that the first week or two was reserved for machine acceptance trials. Since we were already accustomed to doing much of our software checkout on the CSIM simulator, we obtained a pre-release of the 1.14 CSIM, which could simulate both the X-MP CPU and the IOS. We used the 1.14 CSIM extensively, for both COS and IOS development. When we sent a team to Chippewa Falls for final factory trials before shipment, our modified COS 1.12 ran successfully the first time that we tried it. In our environment, with extremely limited test time availability, we simply cannot function without CSIM.

Lest all this sound like everything went smoothly, we had two disastrous beta test sessions just before the X-MP was due to go into production. In both cases, after about half an hour of the beta test, the system destroyed the dataset catalog and crashed. The problem was eventually traced to a last-minute modification that increased the number of SDT entries to 1000. It turns out that field SDQC is only 9 bits wide, which effectively limits COS to no more than 511 SDT entries - we were overflowing this field, with the eventual result of destroying the catalog.

(c) COS 1.14 Conversion

In June BCS started a conversion to COS 1.14. It will take us 29 man-months to convert almost 275,000 lines of local code. Some of the difficulties we are encountering include: massive internal changes between COS 1.12 and COS 1.14; untested code in 1.14; undefined variables in the CFT sections of some 1.14 utilities; and 90 unresolved SPR's for which we had previously submitted corrective code (we expended at least one man-month just converting SPR corrective code).

With the introduction of multi-tasking in COS 1.14, large sections of COS have been rewritten. Unfortunately, we have much local code in some of these areas, especially in the EXP task. Since EXP was extensively rewritten, it would have greatly helped our conversion effort if Cray Research had taken the opportunity to put hooks (user exits) into EXP.

At the start of the conversion, we assembled all of COS, and all the utilities, products, and libraries, then fed these listings to an error analyzer. To our surprise we discovered a total of 52 undefined variables being used in the CFT portions of the following utilities:

BIND	- 6	FLODUMP	- 6
BUILD	- 3	STATS	- 15
CSIM	- 1	TEDI	- 2
DUMP	- 1	UPDATE	- 4
EXTRACT	- 14		

In addition to these blatant bugs, we have also found a handful of problems in COS itself - bugs so obvious that those pieces of code could never have been tested at all. For example, we found two bugs (in 2 lines of code) in the code that computes the number of cylinders for an SSD for the *CONFIG,DVN=SSD parameter file directive. One bug was using an incorrect constant, while the other bug failed to float an integer number before normalizing it.

We have also been experimenting with the Software Tools that were released in 1.14. It appears that they will be extremely difficult to modify. The software tools have their own source management facility, which records program changes in a specially formatted text file (roughly analogous to an UPDATE PL). This internal source file is in turn maintained on an UPDATE PL, resulting in two levels of source management. To modify a software tool, one must first change the program's source, then determine the appropriate changes to the source management tool's internal text file, and finally use UPDATE to change the internal text file. Presumably Cray Research has some automated technique to accomplish all of this. If so, we would like to have access to it.

(d) How Cray Research can help

Cray Research can help BCS in several ways. First, make better releases - no untested code; no undefined variables in CFT code; and better release documentation (although the 1.15 pre-release letter was very good).

Second, Cray Research must greatly increase its concern for and support of binary compatibility of user codes across releases. Although there are some efforts in this direction, a few 'gotcha's seem to slip through each time. Some of the most pernicious impacts involve changes to tables that reside in the user field length. Cray Research should commit to make no changes to the size of such tables, (perhaps all such tables could be doubled in size at one release, to provide room for expansion, and left alone after that), and to never move fields around within such tables. In addition, every such table should have an installation area (including the JCB).

Third, Cray Research should support all products for two COS versions before and after the product is released. In other words, CFT 1.14 should be supported on COS 1.12, 1.13, 1.14, 1.15, and 1.16.

Fourth, Cray Research must address the reliability of COS (and of any successor product). It is no longer an adequate methodology to simply code, test, and fix the bugs that show up. Instead, a design philosophy of fault tolerance and damage containment must be pursued.

Fifth, Cray Research should provide various data center assistance items for their sites. BCS would like to see hooks (user exits) through COS; better scheduling of large jobs (increasingly important as memory sizes grow); a fast, dynamic SSD pre-emption facility; and perhaps most important to us, a super shutdown facility. This super shutdown facility would

allow us to take a system that is loaded with customer jobs, shut it down in such a way that all jobs and their local files are preserved, give the system to a site analyst for a test session, make it appear to that test session that the system is empty (so the analyst can perform deadstarts and not worry about damaging any rolled out or queued jobs), then restore the original production environment.

(iii) Mostyn Lewis (Chevron)

As Chevron Oil Field Research Company (COFRC) took delivery of an X-MP/48 in May 1985, it was necessary to convert to COS 1.14 to support the new hardware (four CPU's, extended addressing, DD-49 disks) and CFT 1.14 also, (to use scatter/gather hardware, for example). Our major problems occurred in CFT, VBS tape support and MVS station support.

CFT had one major and eight critical SPR's. Programs ceased to vectorise; there were "compiler errors" (programs causing compiler aborts); bad code generation and a delay in fixing these problems (in hindsight probably due to COFRC being only a part of a wider arena of CFT victims!).

VBS tapes ceased to work correctly when they were written. This was amazing, as one only had to read what was written to spot this defect. We thought Test and Integration had failed. Two visits from Mendota Heights failed to cure the problem and there was a further compounding of mis-communication when Cray actually believed they had fixed our problem when we were still suffering. A letter from our site analyst was necessary to prompt Cray into a realization of the unsatisfactory support before a final visit found the bugs and cured them. The VBS problem had remained critical for three months.

The MVS Cray station had 12 significant problems. All were fixed by local expertise and included an operation console "hang" problem and another which only allowed a maximum of three active streams (out of 16).

In summary, we believe we observed a breakdown in the communication link from the Cray Region to Field Support to Software Development. Mendota Heights seem to lack the awareness of a customer in trouble. We also noticed how fragile the link to Field Support was, typified by the difficulty our local analysts have in finding the right person ("not answering his phone"). The VBS problem should have been caught by the QA process and really should not have occurred in the first place (apparently an outcome of TQM re-design to support features).

Let's have stability before features, please!

B. COS Interactive: A Developer's Perspective

Bryan Koch (CRI)

1. Overview

This paper briefly describes the functional characteristics of COS interactive, as available through the Station protocol. The applicability of COS interactive to the systems development process is discussed, and the availability of the interactive protocol in various Cray-supplied stations is described.

The majority of the paper is a walk-through of a sample interactive session, using Cray's CSIM simulator.

2. Features of COS Interactive

2.1 Environment

The COS interactive environment is quite similar to the batch environment, so a user familiar with batch should have few if any problems making use of the interactive environment. Control statements are identical in the two modes; procedure (\$PROC) files are usable in either environment; and programs which read from standard input (\$IN) and write to standard output (\$OUT) can be used without modification.

A new class of device types (IA devices) may be associated with dataset names, and the two standard I/O file names \$IN and \$OUT are associated with the interactive station. Reads and writes on \$IN and \$OUT (respectively) cause input lines to be requested from the user and output lines to be written to the user's terminal (via the station). Control statements are also obtained from the user, via the station, when needed.

COS prompts for control statements by issuing a '!'. The standard prompt issued when a user program issues reads on interactive datasets is '?', but this can be changed by inserting a new prompt character in the Job Control Block (JCB).

2.2 Scheduling

No special parameters in the Job Scheduler (JSH) pertain specifically to interactive sessions. Nonetheless, it is possible to some degree to tune the system for interactive processing.

Two levels of scheduling affect interactive users, CPU and memory. Interactive users can be assigned to a specific job class, allowing them to be assigned a different priority than other classes of jobs. (In Mendota Heights, interactive jobs are assigned priority 8, while all other jobs execute at priority 7 or below.) This priority can be used to control the location of interactive jobs in the memory request queue.

2.2.1 Memory Scheduling

All other things being equal, the priority of a job determines the amount of time a job spends in memory (and thus is eligible to execute in the CPU).

When a job which has been suspended for any reason (tape mount request, interactive I/O request, time delay, etc.) is 'resumed' by the operating system, that job receives a one-time priority 'kick', to a level above any other job in the system. This allows jobs which have been waiting for resources (such as tapes) to begin using these resources as quickly as possible.

Interactive jobs receive a priority 'kick' at every control statement and interactive I/O request which results in a suspend. For a system with free memory, this can result in very snappy response times; when jobs must roll out to accommodate the newly-resumed interactive job, the response time depends on the speed of the swap storage and the size of the preempted job(s).

2.2.2 CPU Scheduling

The priority of a job does not control its position in the CPU queue, as this is a function of the time slice, the CPU vs I/O ratio, and the age of the job in the queue.

Standard mechanisms are used to age jobs in the CPU queue, and installations are free to determine the weight that priority plays in this aging. The CPU scheduling algorithm is best described as a 'fair-share' round-robin scheduler. Once in the CPU queue, there is no differentiation between batch and interactive jobs.

2.3 Resource Utilization for Interactive Sessions

Unlike batch jobs, interactive sessions do not begin with a JOB statement. One implication of this is that interactive jobs cannot request any controlled devices, and are bound to use site-wide default limits for other resources.

Thus, interactive jobs cannot typically make use of on-line tapes, because reservations for tape drives must be declared on the JOB statement. Similar situations arise in the use of other controlled devices; sites often make the SSD a controlled device.

Limits for file space, for maximum memory size, and for CPU time, are taken from the site-wide defaults.

3. Data and Control Paths

The Station protocol supports two separate paths from a station user to the interactive session: data and control.

The 'data' path is used for interacting directly with the control statement processor or an application program. It supports ASCII characters, transparent 8-bit characters, as well as a special 'end of file' indicator.

The 'control' path allows the user to bypass normal flow control mechanisms and interact directly with the station and SCP. Control path requests allow the user to:

- * Determine the STATUS of the interactive session. Status information includes the last control statement executed, the CPU status (waiting, executing, suspended), and the amount of CPU time used thus far in the session.
- * SUSPEND or END the session. The user can suspend the session and make use of the facilities of the front-end system, to return at a later time; or the user can end the session, terminating the interactive job.
- * Send an ABORT or ATTENTION interrupt to the currently-executing process. Abort and attention are similar, and many programs and users use them interchangeably. Both may be intercepted by the reprieve mechanism.

Some programs terminate only on ABORT interrupts, and use ATTENTION interrupts to merely suspend their current activity and issue a prompt to the user. CSIM is an example of a program which selectively uses these two flavors of interrupts.

4. Availability

COS interactive is available through the Station protocol with almost all levels of COS used in the Cray community today. Cray-supplied stations which currently support interactive sessions include:

- * IBM Compatible - VM/CMS
- * DEC VAX - VMS and Unix System V
- * Cray IOS
- * Apollo - Aegis

Interactive facilities are under development for the following Stations, with availability scheduled as indicated:

- * IBM Compatible - MVS (1Q86)
- * CDC Cyber - NOS (2Q86)
- * Sun workstations - Unix⁽¹⁾ 4.2BSD (1-2Q86)
- * Other Unix-based products-COS

(1) Unix is a trademark of AT & T

5. Applicability

From the author's perspective as an operating systems developer, one of the primary uses of COS interactive is that it provides a common Cray environment. This environment exists on almost all front-end systems, including the fairly Spartan surroundings of the check-out floor in the manufacturing area.

A common environment means that the developer can learn one environment - the COS interactive environment - and use the Cray almost without regard for the front-end system which provides access to the Cray. This frees the developer to concentrate on solving developmental problems rather than learning yet-another editor or command interface.

The other major use of COS interactive is as a development tool. While some developmental activities - most notably long assemblies and compilations - are more suited to background (batch) processing, many are more appropriate to the interactive environment.

- * Interactive use of Cray systems allows the developer to make use of an iterative development/test cycle.
- * Interactive is often appropriate for simulation and debugging of operating system, library, and user-level codes.
- * Editing, of program source, of procedure libraries, and of data files, is best done interactively, either on the front-end or, in some cases, on the Cray. This can also be used for job creation and submission of background work.

6. Interactive Data Flow

All COS interactive data passes through at least three processes: the Station, the Station Call Processor (SCP) system task, and the user exchange processor (EXP) system task.

6.1 Station Responsibilities

The front-end station is responsible for collecting interactive input from the user, buffering it into record (line) segments, and forwarding these segments to SCP.

Commands, typically denoted by a command-character in the first column of the line, are intercepted and interpreted by the Station rather than being passed as data to SCP. Commands are sent in non-data segments to SCP for processing; some Stations implement local commands as well.

The Station is also responsible for receiving output record (line) segments from SCP and displaying these to the user. As such, the Station is responsible for the form of the displayed data; there is no virtual terminal support in SCP.

6.2 COS System Tasks

Interactive jobs are created by SCP upon receipt of an interactive logon segment from a Station. These jobs are managed by the same mechanisms that control batch jobs - the Job Class Manager.

SCP is responsible for interactive buffer management. When input (from the user) data is received from a Station, it is placed in interactive buffers. When EXP is asked to return the next line of user input, either by an executing program, or when the next control statement is needed, the line is retrieved from the system's interactive buffers.

A similar path operates for interactive output. The user writes a line using standard I/O requests, which are interpreted by EXP. EXP moves the data from the user's buffers to the system interactive buffers. SCP retrieves the data from the interactive buffers and sends it to the appropriate station.

7. Sample Interactive Session

One of the uses of COS interactive is the on-line debugging of operating systems or other programs through simulators or interactive debuggers.

The use of shared time, as opposed to dedicated time, for software development can make developers more productive. It also allows more time for the testing of code, which can result in fewer problems later in the development cycle. The sample session in this section is taken from actual code under development in the Mendota Heights facility.

The code segment we will examine is in EXEC, the portion of COS which operates in monitor mode and handles all exchanges, I/O interrupts, timers, and the like. Because it is in monitor mode, this segment of code would be very difficult to debug using console-oriented (online) tools, and is thus a good example of the power of simulation programs like CSIM.

The remainder of this paper is a walk-through of the way in which a developer might use Cray interactive to debug an operating system modification. The example is presented in dialog form, with sections of the interactive session followed by a discussion of the preceding section.

This example was prepared on a Sun model 2/50 workstation, using a developmental version of the Unix Station running under Unix 4.2BSD. The operating system on the Cray is the bug fix 3 release of COS 1.14.

(The program fragment used in this example is located at the end of the paper.)

```
clem% ias -i
interactive logon - done
```

```
CRAY X-MP SERIAL-201/40      09/24/85
```

```
1.14BF3 COS 1.14 ASSEMBLY DATE 09/17/85
```

```
PD000 - PDN = UPIC ID = BTK ED = 10 OWN =
PD000 - ACCESS COMPLETE
!
```

Discussion

Most interactive stations let the user specify a set of commands to be executed upon interactive logon. Typically, these commands come from a file on the front-end system, and contain account information and other COS control statements to let the user customize the interactive environment. In this session the `ias` command invokes the interactive station, and the `-i` option specifies that an initialization file is to be executed. The author's initialization file contains two directives: an `ACCOUNT` statement, and an `ACCESS` statement.

The system responds to the logon request with the system header (CRAY...), reads and executes the `ACCOUNT` and `ACCESS` statements, then issues the standard control-statement prompt character, '!'. The user is now free to issue any COS control statements, or issue Station commands.

```
! audit,id=btk.
```

```
-----OWN = U9935-----
```

```
PDN      ID      ED
```

```
99COS   BTK   1
START   BTK   1
TESTSSD BTK   1
UPIC    BTK  10
```

```
4 DATASETS, 468 BLOCKS, 239616 WORDS
```

```
! copyf,i=$in,o=param.
```

```
? *INSTALL
```

```
? *END
```

```
? e
```

```
IO048 - COPY OF 2 RECORDS 1 FILES COMPLETED
```

```
! access,dn=cos,pdn=99cos,id=btk.
```

```
PD000 - PDN = 99COS ID = BTK ED = 1 OWN =
```

```
PD000 - ACCESS COMPLETE
```

```
!
```

Discussion

In the example here, the user first issues the `AUDIT` command to determine the names of files created by a separate (batch) run.

`COPYF` is invoked to create a `STARTUP`-style parameter file, needed by `CSIM`. Long or complicated parameter files are typically created using either a local Cray editor (`TEDI`) or on the front-end system, but given the simplicity of the file the user has chosen to create it on-line.

Note that the user is copying from the COS standard input file, `$IN`. When reads are issued on interactive access (`IA`) files, a prompt (`!?`) is sent to the user via the Station, and the interactive session is suspended until the user responds with the requested data.

Note also the use of the special end-of-file command to the station. In the case of the Unix station, this command is `'e'`; other stations have different command conventions.

Finally, the user accesses the COS binary to be simulated.

! csim,t=100.

At 14:04:25 on 09/24/85: Cray CPU/IOS simulation
1.14 CSIM version of 09/13/85 19:50:14

I = \$IN L = \$OUT T = 100.0

? defcpu,x,l,ema
? start cos param

OPSYS = COS OSPAR = PARAM

PARAMETER FILE CONTENTS:

*INSTALL
*END

? dis a 20073b m=e

DIS A 00020073 P EXEC
0020073 006165 030663 071106 007000
0020074 146370 020660 004453 125100
0020075 000004 073201 054226 055202
<etc.>

? bre,l,20074b,m=e
?

Discussion

CSIM, the Cray Simulator, is invoked. It responds with version and parameter information, then issues its prompts the user by reading from \$IN.

The user enters the DEF CPU and START commands, displays memory in the area of the code to be debugged, and sets a breakpoint at the first parcel of the newly-added code.

? run,t=20
DEADSTART

LOGON sent
Segment
0 0000000020040100201000
1 000200000000000000000000
2 0415172462013013430465 COS X.15
3 0300711363106313634065 09/23/85

STARTUP IS PERFORMING AN INSTALL

Breakpoint 1 encountered at P=00020074h
BA=00000000
?x

DIS X X X ANY CPU
P 00020074d A0 00000011 MODES FLAGS
IBA 00000000 A1 00003760 OFF ON OFF ON
ILA 00035000 A2 00000010 MM PCI
XA 3760 VL 100 A3 00006165 ICM MCU
DBA 00000000 A4 00047115 IFP FPE
DLA 04000000 A5 0005343 IUM ORE
B00 00020074b A6 00004453 IMM PRE
A7 00004303 SEI ME
BDM IOI

S0 00000000000000000000000000000000 FPS EEX
S1 00000000000000000000000000000000 WS NEX
S2 02000000000000000000000000000000 IOR DL
S3 00000000000000000000000000000003 EMA ICP
S4 000000104000000000001000 SVL
S5 00000000000000000000000000000001
S6 00000000000000000000000000000002
S7 10000000000000000000000000000000
VM 00000000000000000000000000000000
PROCESSOR = 0 CLUSTER = 1 PS = 0
ERROR TYPE = NONE VNU = 0
CHIP SLCT = 00 BANK = 00
READ MODE = I/O SYNDROME = 000
?

Discussion

The RUN directive begins simulation of the newly-loaded binary image. The time limit for this section of the simulation will be twenty seconds of real CPU time. The built-in station within CSIM sends a LOGON segment to the simulated system, and the simulated STARTUP replies with the message STARTUP IS PERFORMING AN INSTALL. Eventually, the breakpoint set earlier is encountered.

The user issues the CSIM Station command 'X' to examine the currently-executing exchange package in the simulated system.

?step

TRACE CPU-0 P=00020074d BA=00000000 S1 4,A5
Result register S1 = 000000104000000000001000

?step

TRACE CPU-0 P=00020075b BA=00000000 S2 SR0
Result register S2 = 100000000004000000000000

?step

TRACE CPU- 0 P=00020075c BA=00000000 S2 S2<26
Result register S2=000100000000000000000000

?step

TRACE CPU- 0 P=00020075d BA=00000000 S2 S2>76
Result register S2 = 000000000000000000000000

?step

TRACE CPU- 0 P=00020076a BA=00000000 S0 S1S2
Result register S0 = 000000104000000000001000

?a

DIS A 00020073 P EXEC
0020073 006165 030663 071106 007000
0020074 146370 020660 004453 125100
0020075 000004 073201 054226 055202
<etc.>

? 20074d=126100,m=e

? a

DIS A 00020073 P EXEC
0020073 006165 030663 071106 007000
0020074 146370 020660 004453 126100
0020075 000004 073201 054226 055202
<etc.>

? p=20074d,m=e

?

Discussion

The user begins stepping through the code, one instruction at a time.

By the time the last **step** is reached, something has gone wrong. The result in S0 register should be the difference between the actual CPU number and that of the Guest Operating System (GOS). The number in S0 is clearly wrong since CPU numbers range from 0-3.

Analysis of the code locates the problem. The instruction at 20074b loads A6 with the base address of the Guest Operating System table, while the next instruction uses A5 as the index register. One or the other is wrong, as they should be identical. A quick scan of the following code shows that A6 is used as a scratch register, so the developer decides to change the GETF to use A6.

The instruction in simulated memory is modified, and the P-register is reset to the beginning of the code segment in question.

```
?step
TRACE CPU-0 P=00020074d BA=00000000 S1 4,A6
Result register S1 = 000000000000000000000000
?step
TRACE CPU-0 P=00020075b BA=00000000 S2 SR0
Result register S2 = 100000000004000000000000
?step
TRACE CPU-0 P=00020075c BA=00000000 S2 S2<26
Result register S2 = 000100000000000000000000
?step
TRACE CPU-0 P=00020075d BA=00000000 S2 S2>76
Result register S2 = 000000000000000000000000
?step
TRACE CPU-0 P=00020076a BA=00000000 S0 S1S2
Result register S0 = 000000000000000000000000
? step
TRACE CPU-0 P=00020076b BA=00000000 JSN 20111a
? step
TRACE CPU-0 P=00020076d BA=00000000 A6 S1
Result register A6 = 00000000 <etc.>
? step
TRACE CPU-0 P=00020105c BA=00000000 A0 A6-A4
Result register A0 = 77777700
?
```

Discussion

The developer steps through the remainder of the code to ensure that there are no other bugs in the code segment. Note that after the initial S-register comparison (P=20076a), the correct (or at least reasonable) value is now returned.

? run

DEVICE DD-A1-20 - NO ENGINEERING FLAW
TABLE FOUND

ENTER GO TO CONTINUE STARTUP
SKIP TO CONTINUE W/O FURTHER WARNINGS
? SKIP

```
STARTUP SELECTED CREATION OF THE
$DSC-EXTENSION.
THE $DSC-EXTENSION WAS CREATED AND SAVED
SUCCESSFULLY.
THE $DSC-EXTENSION IS 00% FULL.
THE $DSC-EXTENSION WAS RECOVERED AND
VALIDATED SUCCESSFULLY.
CREATING NEW EDITION OF THE SYSTEM
DIRECTORY
```

Simulation time limit exceeded.
? run,t=40

```
*** SIMULATION COMPLETE ***
? end
! q
BYE
```

Discussion

Having successfully tested the developmental code, the programmer elects to let STARTUP run to completion. The initial time limit of 20 seconds is insufficient, but the programmer allocates more time and the simulation then runs to completion.

The END directive terminates the CSIM session, and the user then issues the Station logoff command (in this case 'q').

8. Conclusions

COS interactive, as provided for by the interactive Stations, fulfills an important role in the software development process in Cray's Mendota Heights facility. Large assemblies and compilations are performed in the background, while interactive sessions are used in the debugging, testing and validation of new operating system, library, and applications-level code.

Interactive sessions have available the same command set used in the batch environment, though some limitations are imposed by the lack of interactive JOB statements in the area of resource utilization.

A common scheduler is used for both batch and interactive sessions. Some tuning is possible through the use of the Job Class Manager. Response time to interactive sessions is largely controlled by the amount of free memory, and the size of jobs needing to roll out to free memory for interactive sessions.

Interactive Session Example

```
20073b 030663      A6      A6+A3      A6 <= STT addr
      c 071106      S1      A6
      d 007 00031476a R      TACT      ready the task
```

----- new code segment -----

* Check if running in the GOS CPU.

```
20074b 0206 00004453 A6      B@GOS+LH@GOS
      d <opdef>      GETF,S1 S7,GOSCPU,A5  GOS CPU number
20075b <macro>      GETSR0 PN,S2      Current CPU
20076a <macro.     $IF      S1,EQ,S2      If in the GOS CPU
      d 023610      A6      S1      Find processor to interrupt
20077a <macro>      GENPBM S3,A6      GOS processor
20100c 1202 00002016 S2      PSMEXEC,0     EXEC CPUs
20101a 1204 00002020 S4      PSMSTP,0      STP CPUs
      c 051224      S2      S2!S4
      d 045023      S0      #S3&S2      EXEC or STP CPUs not
assigned
20102a <macro>      $IF      SZ      If COS needs to be interrupted
      c 1202 00002014 S2      PSMIDLE,0  IDLE CPUs
20103a 1204 00002022 S4      PSMUSER,0  USER CPUs
      c 1205 00002026 S5      PSMDOWN,0  DOWN CPUs
20104a 1006 00035013 A6      XEND+HIGHCPUN,0 Max CPU number
      c 051224      S2      S2!S4      IDLE and USER CPUs
      d 051535      S5      S3!S5      DOWN and GOS CPUs
20105a 045225      S2      #S5&S2     All CPUs eligible to interrupt
      b 027420      A4      ZS2      Processor number or 64
      c <macro>      $IF      A4,LE,A6     If a valid COS CPU to
interrupt
20106b <macro>      SETIP      PN=A4,SCR=A6,ERROR=$STOP112
20111a <macro>      $ENDIF
      a <macro>      $ENDIF
      a <macro>      $ENDIF
```

C. OSSIC Report

David Lexton (ULCC)

The Operating Systems committee had met on Monday 30th September. Don Mason gave a report on COS and UNIX. The situation on User Requirements was then considered in some detail. On behalf of CRI, Don Mason had responded (to Steve Niver, chair of the User Requirements committee) to those forwarded to them from the first ballot. OSSIC needs to consider the responses to operating systems requirements but as no one on the committee had seen them prior to 30 September, this was not yet possible, as far as all the requirements were concerned. However, in relation to user exits, system tuning and installation areas, the OSSIC Chair had collected detailed material from a number of sites in June and sent that to CRI. In spite of this Don Mason had requested more discussion on those topics. The committee felt very strongly that a general approach to these questions was urgently required. To facilitate this, Conrad Kimball, Jim Sherin and Dean Smith agreed to interact directly on behalf of OSSIC with Don Mason's nominees on these matters. CRI have asked for input from OSSIC on two UNIX issues, namely the operator interface and batch processing requirements. User requirements to be submitted to URC for inclusion in the next ballot were discussed.

The committee appointed a new chair Ray Benoit (CID), with Dave Lexton (ULCC) as deputy-chair. The other members of the committee are Conrad Kimball (Boeing), Jim Sherin (Westinghouse), Lothar Wollschlager (KFA), Claus Hilberg (ECMWF), Mostyn Lewis (Chevron), Larry Yaeger (Digital Productions), Charles Slocomb (LANL) and Don Mason (CRI).

The Committee discussed the relation between the SIC and CUG sessions, without reaching any conclusions. It was thought desirable to have some overlap between the areas covered by different SICs.

MICROTASKING PANEL SESSION

Mary Zosel

Lawrence Livermore National Laboratory
Livermore, California

Frank Bobrowicz, LANL, Mike Booth and Lisa Krause, CRI, and Rob Strout, LLNL made up a panel for discussion of the microtasking approach to using multiprocessors. Lisa Krause discussed the status and plans of the microtasking preprocessor at CRI. It is currently being prepared for beta release in system 1.15. There have been several bug fixes made to the original prototype version. Changes to allow separate compilation have been made. There are also plans to support a new form of the guard directive: `CDIR$ GUARD n` where `n` is treated as $(n \bmod 64)$ to generate a lock number.

Rob Strout described what was done to get the preprocessor and microtasking library up and running on the NLSS system at LLNL, and in general what must be done to move to a different operating system. The changes are minimal. The target system and compilers must support stack based code generation. Minor preprocessor changes were required to make up for different library support. The library change for NLSS required only a change to the method used to give up idle processors.

Mike Booth led a discussion about interaction between multitasking and microtasking and how he could see them working together in the future. He discussed some ideas about making use of the hardware performance monitor to make operating system scheduling decisions to avoid assigning processor resources to microtasking slaves which were hanging on a semaphore. Since there are other customer uses for the performance monitor, the system would have used this information in a way which did not preempt other usage.

Mike also answered general questions about how microtasking works, what context switching is done (none) and why/where a user might want to use microtasking. Microtasking introduces use of multiple processors for loops with very low execution time overhead and a simple syntactic directive. It does introduce extra cost in terms of code size, because some code is duplicated in the system. It also may slow down overall system throughput, if the operating system is not carefully tuned, because it leaves slave processors hanging on semaphores when the applications is in a monoprocessor portion of the code.

PERFORMANCE AND EVALUATION SPECIAL INTEREST COMMITTEE (PESIC)

Mostyn Lewis

Chevron Oil Field Research Company
La Habra, CA

PESIC was formed at the 85' Fall CUG in Montréal. It comprises the original three workshops on:

- Performance (Ann Cowley, NCAR)
- Optimization (Jacqueline Goirand, CIFRAM)
- I/O (Mostyn Lewis, COFRC)

Also to be included in future PESIC sessions are topics on:

- Benchmarking
- Workload Analysis
- System Tuning

No demarcation on hardware is implied; all Cray machines 1s, Xs, 2s, (Ys and 3s) are to be included.

An important new topic to be embraced is software reliability. Any suggestions in this area are welcomed. PESIC would like hard facts and philosophy.

Traditionally, experiences with new software and hardware have been presented such as relate to the SSD and DD-49's and new I/O software. We would like to continue this pioneering trend and invite anyone to contribute. (How about 3480 tape experiences, DD-39 experiences, UNIX related experiences...?).

If you have something to contribute at the next conference in Seattle during the spring of 1986 where the theme is UNIX and you are not sure if it fits into any of the obvious SIC's, give me a call -- PESIC is your safety net and is likely to be able to accommodate you.

I/O WORKSHOP

Mostyn Lewis

Chevron Oil Field Research
La Habra, CA

The I/O workshop had four speakers covering benchmarking, applications, new Cray I/O features and new hardware experiences. The breadth of the coverage was an apt precursor to the newly-formed PESIC (Performance and Evaluation Special Interest Committee) of which the I/O workshop is one constituent. For more information on PESIC, please see elsewhere in these proceedings.

BENCHMARKING CRAY'S X-MP AND SSD

Christopher Holl

Boeing Computer Services
Bellevue, WA

Starting in 1984, Boeing Computer Services benchmarked several X-MP / SSD systems to learn more about the model 24 and 128 Meg SSD which were installed in March of 1985. The tests conducted over the year revealed much about the behavior of the mainframe compared to the Cray 1-S. Runs with the SSD measured the characteristics of this very high speed storage device. In addition to jobs designed for the benchmarks, many applications were run with and without the SSD to judge the improvement over the 1-S, and observe the impact of the SSD on production work.

This presentation covers a general description of the benchmark tools, followed by the results of CPU and SSD performance testing.

I. GOALS

i. CPU Speed Factor

The first goal was to obtain a ratio of the average CPU seconds used by each mainframe while executing a single job. If the number of 1-S CPU seconds used is known, this ratio, or "speed factor" can be used to estimate how many CPU seconds will be used on the X-MP.

ii. Throughput

The second was to determine the throughput of the X-MP relative to the Cray 1-S, while processing a "typical" Boeing workload.

iii. I/O (disk and SSD)

The final goal was to measure the disk I/O, and the speed of the SSD.

II. HARDWARE DIFFERENCES

i. CPU

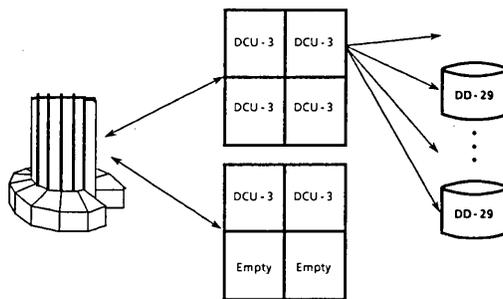
There are three major differences between a 1-S CPU and an X-MP CPU. The clock cycle has been reduced from 12.5 nanoseconds on the 1-S to 9.5 nanoseconds on the X-MP; the number of paths to memory (per CPU) has increased from one to three; and later X-MPs (including Boeing's) have hardware scatter/gather and compressed index instructions, while the 1-S must use software for these operations. (The vector floating point multiply unit can also be used as a second logical unit in the X-MP, but this option was not used for the benchmarks.)

ii. Memory

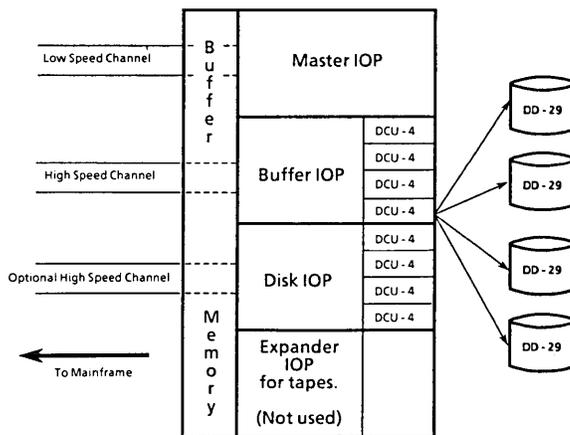
The amount of central memory (for Boeing's machines) has increased from two million words on the 1-S, to four million words on the X-MP.

iii I/O

Boeing's Cray 1-S has DCU-3 disk controllers, while the X-MP has an IOS with DCU-4 controllers. A 128 million word Solidstate Storage Device (SSD) was installed with the X-MP. Figure 1 displays the two configurations.



Cray 1-S Disk Configuration



IOS Configuration

Figure 1: Boeing's I/O Configurations

III. TOOLS

i. Boeing and commercial jobs:

The first category of tools consists of jobs obtained from Boeing Computer Services Engineering Technology Applications group (ETA). The six types of ETA tests are summarized in Table 1. Additional tests (not listed here) were solicited from commercial customers and the Boeing customer for a second benchmark. These tests consisted of structural, thermal, reservoir, fluid dynamics, and dynamic analysis codes.

<u>Designation</u>	<u>Description</u>
FFT1	Fast Fourier Transfer, certification case.
FFT2	Test case, array size: 20482.
FFT3	Test case, array size: 40962.
FFT4	Test case, array size: 81922.
LES1	Linear Equation Solver: Envelope Factorization.
MDL4	NASTRAN* model of gas generator for off-shore platform.
SAM0	Job to assemble SAMECS (Boeing structure code) modules.
SAM1	SAMECS data case.
SAM2	SAMECS data case.
TARIO	SAMECS / ATLAS I / O routine.
VIP	Vector Implicit Program,** Reservoir simulation.

* By McNeal - Schwindler Corporation.
** By Nolen & Associates (with permission).

Table 1: Engineering applications tests

ii. Programs designed for the benchmark.

The second category consists of three jobs developed by Boeing Computer Services Cray Technology. These jobs are listed in Table 2.

<u>Designation</u>	<u>Description</u>
FLOPS1	N = 500. Solve $Ax = I$ for x . Data in memory.
FLOPS2	N = 500. Amount of I/O = 125,754 words.
FLOPS3	N = 1000. Data in memory.
FLOPS4	N = 2500. Amount of I/O = 3,128,784 words.
FLOPS5	N = 5000. Amount of I/O = 12,507,630 words.
IOSTAT	Job to measure transfer rate and overhead of an I/O device.
SORT	Sort-merge test.

Table 2: Boeing Technology tests

iii. QBM / QBMCPU

The third category consists of a benchmark tool named Quick BenchMark (QBM) because of its ability to determine the relative capacity of a configuration in approximately ten minutes. A complete description of QBM is outside the scope and intent of this presentation. The function of QBM is to duplicate a ten minute subset of a workload running on a *base* system. This workload can then be run on any *target* system. If the QBM load processes in ten minutes on the target computer, it has the same capacity as the base computer. If the load processes in more than ten minutes, the target system has less capacity than the base system. If the load finishes in less time, the target system has greater capacity. The QBM load can be scaled up or down until the job takes ten minutes to complete. The relative capacity of the target machine *for running the base machine's workload* is the load factor used to scale the base load so it processes in ten minutes on the target machine. This is a simplified overview, but the process will give results such as 1 Cray X-MP = 2.5 Cray 1-S'.

A special QBM job was also used to measure the speed factor. This job, called QBMCPU, performs 11 different types of computation, as described in table 3. A *V* indicates that the exercise is vectorizable. Each computation is weighted by a best guess approximation to reflect its percentage of the total Boeing workload. Table 4

	<u>CPU COMPUTATION DESCRIPTION</u>	<u>WEIGHT</u>
1.	3-Dimensional Average Flux Code	.05 V
2.	2-Dimensional Linear Algebra	.15 V
3.	1-Dimensional Linear Algebra	.15 V
4.	100 Element Random Scatter	.05
5.	100 Element Random Gather	.05
6.	100 Calls to Function With 1 Argument	.03
7.	100 Calls to Function With 6 Arguments	.03
8.	100 Calls to Library Routines: SINE, COSINE, EXPONENTIAL	.10 V
9.	One Call to Linear Code	.14
10.	One Call to Shell Sort of 2 Word Records	.05
11.	100 Calls to Formatted I/O Loop	<u>.20</u>
		1.00

Table 3: QBMCPU tests

shows QBMCPU to be 45% vector code. The exercises were calibrated on Boeing's Cray 1-S (running the COS 1.10 operating system) so the program could print speed factors directly. QBMCPU was then run on a Cray 1-S and an X-MP in Mendota Heights, both running COS 1.12. Speed factors for each exercise, and the weighted overall factor, were obtained on the X-MP, and "normalized" by dividing by the speed factors found on the Mendota 1-S. This adjustment was necessary since the tests were run on a different release of COS than that on which the program was calibrated.

V. RESULTS

i. CPU

Boeing and Commercial Jobs:

Each of the test jobs was run on Boeing's Cray 1-S (and in some cases Cray's 1-S in Mendota Heights) and again on an X-MP. The CPU times for the major job steps were obtained and compared. Each pair of times provides a ratio of CPU seconds used by each mainframe for a given task. The hardware changes already mentioned affect this ratio. First of all, it is reasonable to suppose that the ratio of CPU time

used by a job would be the ratio of the clock speeds: $12.5 / 9.5 = 1.32$, or that a job that took 12.5 CPU seconds on a 1-S would take 9.5 CPU seconds on an X-MP. The other changes come into consideration however. The 1-S has only one path to memory for fetching and storing data. The X-MP has four paths to memory for each CPU: one for storing data, two for fetching data, and one for I/O. This allows it to perform some operations with less contention for memory access than a 1-S. For example, in the vector operation $A = B + C$, the system performs a fetch for each element of vectors B and C , adds the values, and stores the result in the corresponding element of A . The fetch of b_i and c_i can take place at the same time. Once the sum is calculated, the store of a_i can take place at the same time as the fetch of b_{i+n} and c_{i+n} , etc. This advantage only becomes appreciable when processing vectors. The longer the vector length, the better the X-MP performs. This means some calculations can have a large increase in performance. Observations showed improvements of up to twice the speed of a 1-S. The theoretical maximum speed factor for a function designed to take full advantage of the X-MP hardware would be **3 (memory paths) X 1.32 (CPU factor) = 3.96**.

Influencing the low end of performance is the fact that even though the CPU (i.e. clock cycle time) is faster, *the memory access time is not*. It takes the same length of time to fetch an element from memory into the CPU on an X-MP as it does on a 1-S. All benchmark tests were performed using CFT 1.11 to allow direct comparison with tests executed on Boeing's 1-S. Optimization for memory timing done by CFT 1.11 uses the timing for a 1-S: 9 clock cycles of work are scheduled after a memory request. To wait the same length of time, the faster X-MP must wait 12 cycles. This means at least 3 cycles are wasted for every memory request using binaries generated by CFT 1.11. Memory access bound jobs may show little or no improvement. This was confirmed by the software scatter and gather tests, which showed almost no performance gain. The range of the speed factor for a single CPU with its associated paths to memory is then from 1.0 to 3.96.

The CFT 1.13 compiler allows the user to specify on which type of mainframe the program will execute. When the compiler generates binaries for the X-MP it will try to schedule 12 clock cycles worth of operations during the time that would be idle due to a memory access. If CFT 1.13 can do this so there are no more idle cycles than there would be on a 1-S, the minimum improvement should be 1.3. For CAL programs, the programmer must optimize for timing considerations.

<u>Job</u>	<u>Average of 1-S times</u>	<u>Average of X-MP times</u>	<u>Ratio</u>
FFT1	4.4183	3.2750	1.35
FFT2	7.2460	4.6115	1.57
FFT3	31.2264	20.1143	1.55
FFT4	136.3585	89.6515	1.52
LES1	187.8939	147.2305	1.28
MDL4	1965.9437	1275.2191	1.54
SAM0	27.9868	18.4480	1.52
SAM1	96.1758	77.6475	1.24
SAM2	334.1038	265.0155	1.26
TARIO	0.2487	0.1900	1.31
VIP	554.9596	280.0902	1.98
FLOPS1	2.2862	1.5981	1.43
FLOPS2	2.3857	1.6629	1.43
FLOPS3	11.0417	7.0957	1.56
FLOPS4	113.1866	64.3849	1.76
FLOPS5	751.0158	395.4812	1.90
SORT	23.6077	15.7917	1.49
		Average Ratio	1.51

Table 4: Benchmark Job CPU Times and Ratios

Engineering and Technology Jobs:

The speed factors obtained for these jobs are listed in table 4. These factors were averaged to give the overall mean speed factor *for the benchmark jobs*. In production, the overall speed factor will be determined by the ratio of scalar to vector work being performed by the user base. We had no way to measure this ratio on the 1-S, so we must assume the benchmark jobs are representative of the workmix. The X-MP has hardware registers that accumulate performance statistics for a job. Note that the factor for SAM0 is close to the average. SAM0 is a setup job for the SAMECS tests, and consists of 9 compile, load (without execution), and save steps. MDL4 also has a speed factor very close the the average. To show the distribution,

the speed factors for the jobs listed in table 4 are plotted in figure 2 . (IOSTAT was not used as a speed factor test.) A second benchmark was conducted with the additional jobs and the distribution for these is plotted in figure 3. The tests which were able to compile under CFT 1.14 were run with hardware scatter/gather, and the distribution for these is shown in figure 4.

QBM:

The results of the QBMCPU kernels are displayed in Table 5. During the course of QBMCPU testing, several copies of the program were run together. The speed factors produced by these runs varied more than expected, which led to some investigation. QBMCPU was run 8 times on an X-MP. Some of these jobs were run

TEST	SMALLEST	LARGEST	AVERAGE	% DIFF
1.	1.3171	1.3629	1.3497	3.48
2.	1.6171	1.9074	1.8249	17.95
3.	1.9677	2.3155	2.2343	17.67
4.	1.1458	1.1806	1.1725	3.03
5.	1.0739	1.0969	1.0908	2.14
6.	1.2127	1.2588	1.2480	3.80
7.	1.1658	1.2108	1.2000	3.86
8.	1.4311	1.4636	1.4562	2.27
9.	1.2087	1.2463	1.2374	3.11
10.	1.1107	1.1499	1.1407	3.53
11.	1.2532	1.2938	1.2859	3.24
Overall			1.500	4.95

Table 5: QBMCPU Speed Factor Results

single stream (alone in the computer) and some were run together (up to four at a time). Each job took 10 measurements. This gave 80 samples for each type of computation. The percent difference is $(\text{largest} - \text{smallest}) \div (\text{smallest}) \times 100$. Of the 11 CPU tests, two showed a variation in the amount of CPU-seconds-used in excess of 10%. Test 2 executed the following code in a loop:

```

DO 2 J = 1,5
  DO 1 I = 1,100
    A(J,I) = A(J,I) + R1(I) * C(J,I) + D(I,J)
  1 CONTINUE
2 CONTINUE

```

The time required to perform this double loop varied by 18%.

Figure 2:
FIRST BENCHMARK DATA
(COS 1.12 - CFT 1.09)

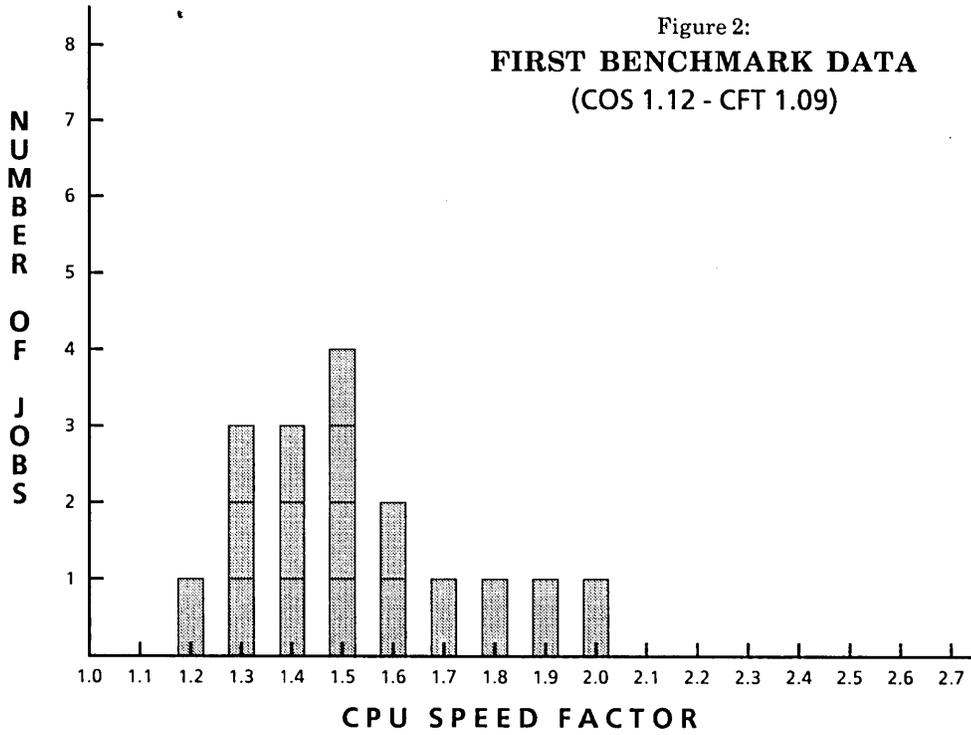


Figure 3:
SECOND BENCHMARK DATA
(COS 1.12 - CFT 1.11)

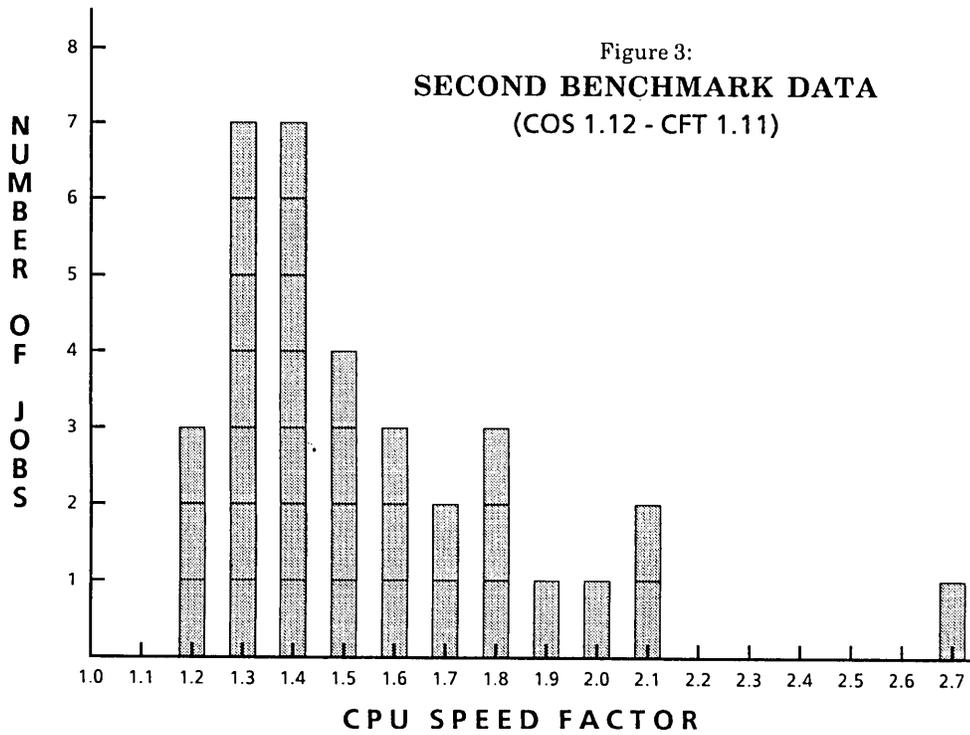
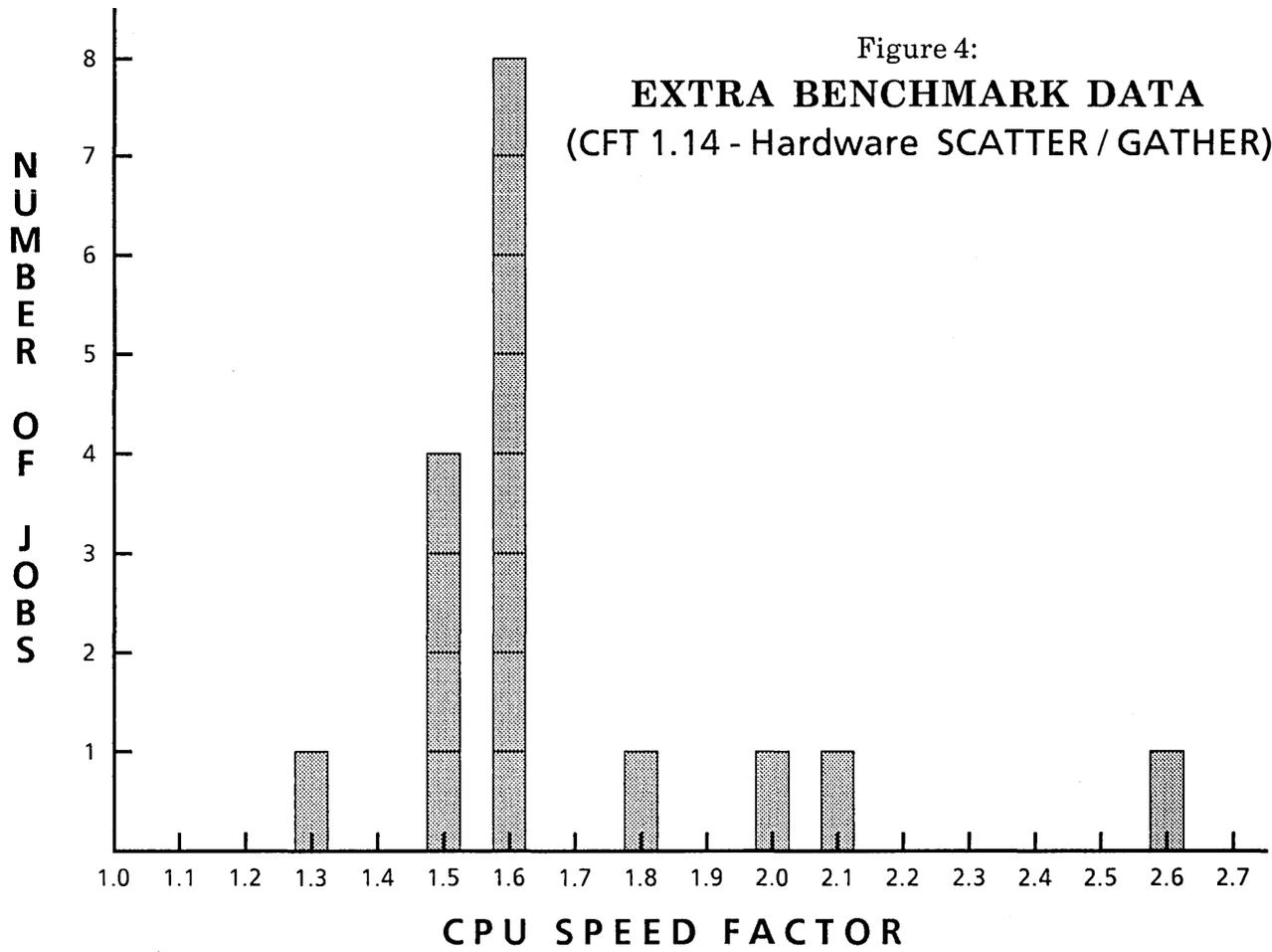


Figure 4:
EXTRA BENCHMARK DATA
(CFT 1.14 - Hardware SCATTER / GATHER)



Test 3 was a similar loop:

```
DO 1 I = 1,100
  H(I) = H(I) + R1(I) * PH(I) + D1(I)
1 CONTINUE
```

The CPU times for this test also varied by more than 17%. The fetching and storing of elements was causing one CPU to wait for access to a bank of memory, while the other CPU was reading from, or writing to, that bank. The tests were changed slightly in an attempt to create a worst case. Test 2 was shortened to performed two FETCHes and a STORE. This is the maximum number of memory accesses the hardware can support at one time, and it was hoped this would intensify the symptom. Test 2 became:

$$A(J, I) = C(J, I) + D(I, J)$$

Test 3 was changed to three FETCHes and a STORE which produces out-of-stride memory accesses:

$$H(I) = R1(I) * PH(I) + D1(I)$$

The number of times each expression was executed was increased for each test, to maintain the proper weighting factors. This modified program was called QBMCPU2. This job was then run the same number of times that QBMCPU had been run. Changing the expressions did indeed affect the ability of the tests to use a consistent number of CPU seconds. Test 2 now varied by 19% and test 3 by as much as 26%. A complete summary of the QBMCPU2 results are listed in Table 6. Cray

TEST	SMALLEST	LARGEST	AVERAGE	% DIFF
1.	1.3104	1.3627	1.3493	4.00
2.	1.5185	1.8115	1.7046	19.30
3.	2.1452	2.7068	2.5571	26.18
4.	1.1421	1.1802	1.1721	3.33
5.	1.0858	1.0979	1.0949	1.12
6.	1.1963	1.2482	1.2389	4.33
7.	1.1634	1.2168	1.2107	4.59
8.	1.4464	1.4638	1.4597	1.20
9.	1.2820	1.3011	1.2964	1.49
10.	1.1007	1.1520	1.1431	4.66
11.	1.2655	1.2943	1.2867	2.27
Overall			1.5392	7.48

Table 6: QBMCPU2 Speed Factor Results

Research was asked to respond to this inconsistency with some quantitative explanation. They were given the source to QBMCPU with instructions for use, and obtained approximately 17% variability for some of the kernels. Although they recognize that some variability exists, they have offered no recommendation for controlling the problem.

In preparation for the test of the 4-CPU Cray, QBMCPU was enhanced to include four additional exercises (see table 7). These new tests were added to aggravate the

	<u>CPU COMPUTATION DESCRIPTION</u>	<u>WEIGHT</u>
12.	FETCH-FETCH-STORE double loop	.05 V
13.	3000 word storage move	.05
14.	100 CALLs to a function with FLOW TRACE enabled	.01
15.	100 CALLs of a 9-word BUFFER OUT	.03

Table 7: Enhanced QBMCPU on 4-CPU X-MP

CPU variability problem, and determine experimentally how widely the times could vary. (These weights can no longer be interpreted as percentages, since the sum of all weights now totals 1.14.) QBMCPU was recalibrated and run 15 times on the 4 CPU Cray X-MP, giving 150 measurements per kernel. The results for both speed factors and variability are presented in Table 8. Notice the high ratios for the SCATTER and GATHER tests (4 and 5). The 4 - CPU model had the hardware scatter/gather feature, while the X-MPs in previous tests did not. The X-MP that Boeing purchased has this feature. Since these operations are so much faster than the other types of CPU tests, even small weighting coefficients will cause a large difference in the overall CPU speed factor. In previous tests, the speed factor has been determined to be 1.52, while the new CPU gives 2.55. How much this affects the other benchmark tests is unknown, but it is possible the potential of the CPU has been underestimated, and the throughput may be greater than measured. As mentioned before, the actual performance obtained will depend on the work being performed. It is not possible to determine how much of the Boeing workload consists of scatter/gather operations. The variation in CPU time is greater than on the 2 CPU X-MP, which is to be expected, since twice as many CPUs are trying to access memory through twice as many CPU-to-memory paths. The overall CPU variability is almost 20%, with a large standard deviation.

<u>TEST</u>	<u>SPEED FACTOR</u>	<u>% DIFFERENCE</u>
1.	1.36	6.17
2.	1.84	34.49
3.	2.05	26.11
4.	11.21	38.43
5.	13.94	38.23
6.	1.17	5.26
7.	1.20	7.15
8.	1.39	4.16
9.	1.52	4.11
10.	2.03	6.23
11.	1.25	4.14
12.	1.73	33.10
13.	1.84	38.90
14.	1.31	8.69
15.	1.10	3.43
Overall	2.55	19.84

Table 8: Enhanced QBMCPU on 4-CPU X-MP

THROUGHPUT RESULTS

QBM was run on Boeing's 1-S and on several X-MPs. The last X-MP benchmark was conducted on Boeing's model 24 after installation at the Bellevue datacenter. The operating system was COS 1.12, with CFT 1.11. These tests showed that the X-MP would process 3.85 times the Boeing Computer Services' Cray 1-S workload.

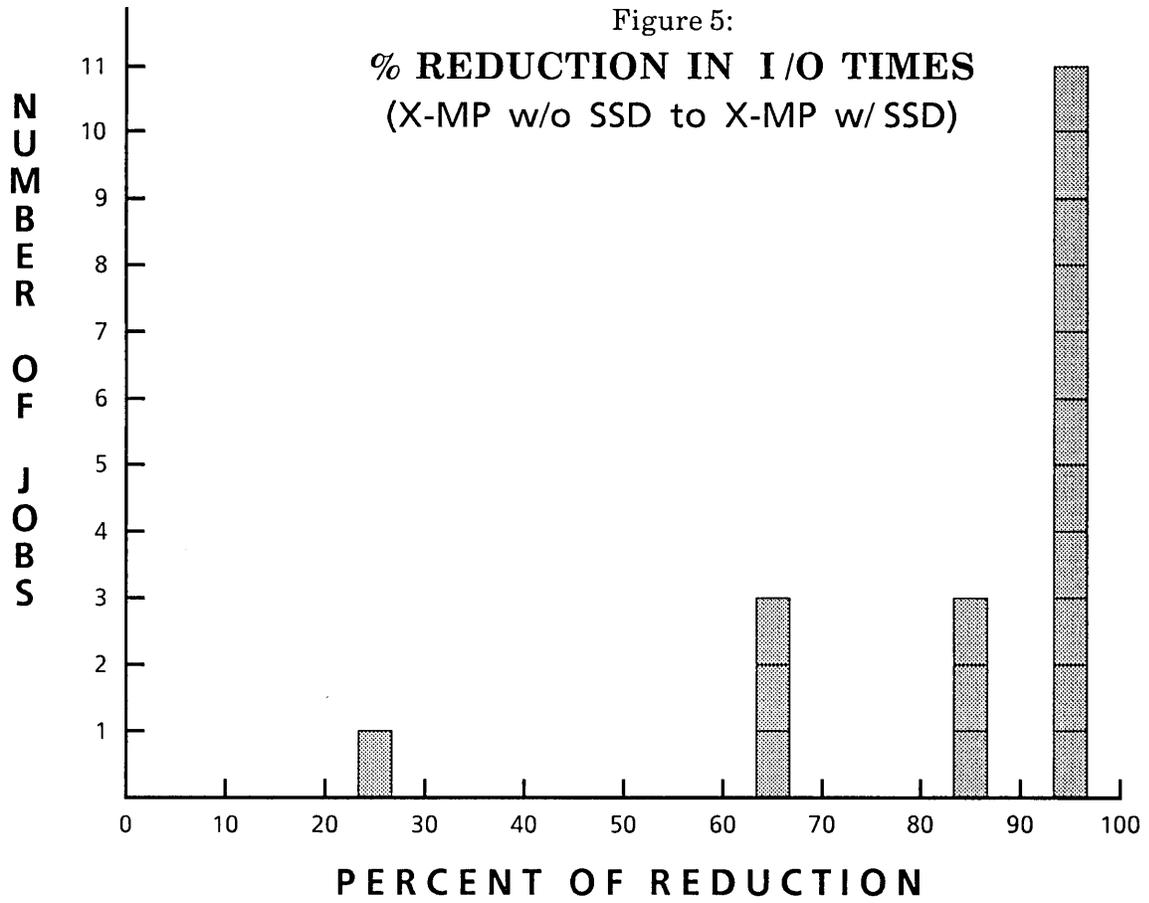
SSD

Each of the benchmark jobs was run once with all datasets resident on disk, and a second time with scratch datasets assigned to the SSD. The CPU seconds were subtracted from the wall clock time to give an estimate of I/O time. Since each job was run alone in the machine this is a reasonable approximation. The percentage of reduction for the jobs is graphed in figure 5. Only the jobs with a fair amount of I/O (greater than 10 seconds with SSD) are represented. Almost all jobs showed a tremendous reduction in I/O time, the best case being shown below.

IOSTAT showed how the transfer rate of the SSD (and Buffer Memory) is affected by the buffer size, and that the default buffer size of 4 blocks is too small. Boeing has changed the default size to 40 blocks (for SSD and BMR datasets) in order to realize

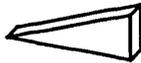
Figure 5:

% REDUCTION IN I/O TIMES
(X-MP w/o SSD to X-MP w/ SSD)



BEST EXAMPLE
Computational Fluid Dynamics

3-D Parabolized Navier-Stokes
10° Wedge with corner flow



<u>System</u>	<u>Wall Clock Time</u>	<u>CPU seconds</u>	<u>I/O Time</u>
1-S	1:53:46	4928	31:38
X-MP	1:25:10	3752	22:38
W/SSD	1:03:05	3750	00:35

approximately 75% of the effective transfer rate. Figures 6 through 9 show various I/O plots for three different sized SSDs, and Buffer Memory.

V. CONCLUSIONS

Although the data has more to reveal, some basic conclusions have been drawn from the benchmarks:

- . CPU performance is very dependent on type of work.
- . The amount of CPU seconds used can vary.
- . Throughput for Boeing is 3.85 times a 1-S.
- . The SSD can reduce I/O time by 90%.
- . SSD buffer size needs to be larger than disk buffer size.

The hardware performance registers on the X-MP can be used to obtain a more detailed profile of the type of work being done. This will allow for more accurate benchmarks to be done on future vector processing computers.

Figure 6:
SEQUENTIAL SSD TRANSFER RATE

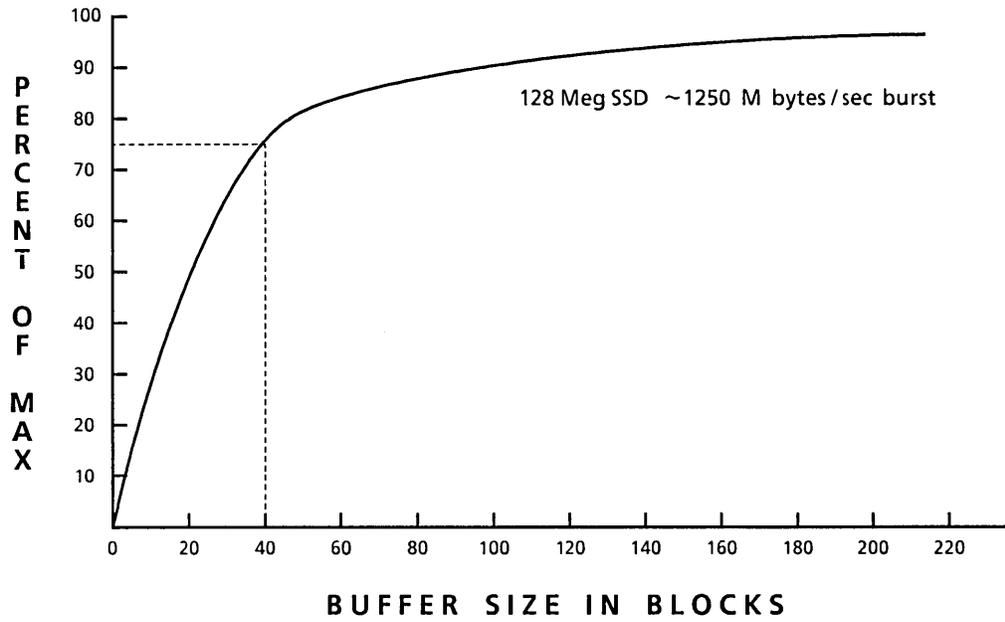
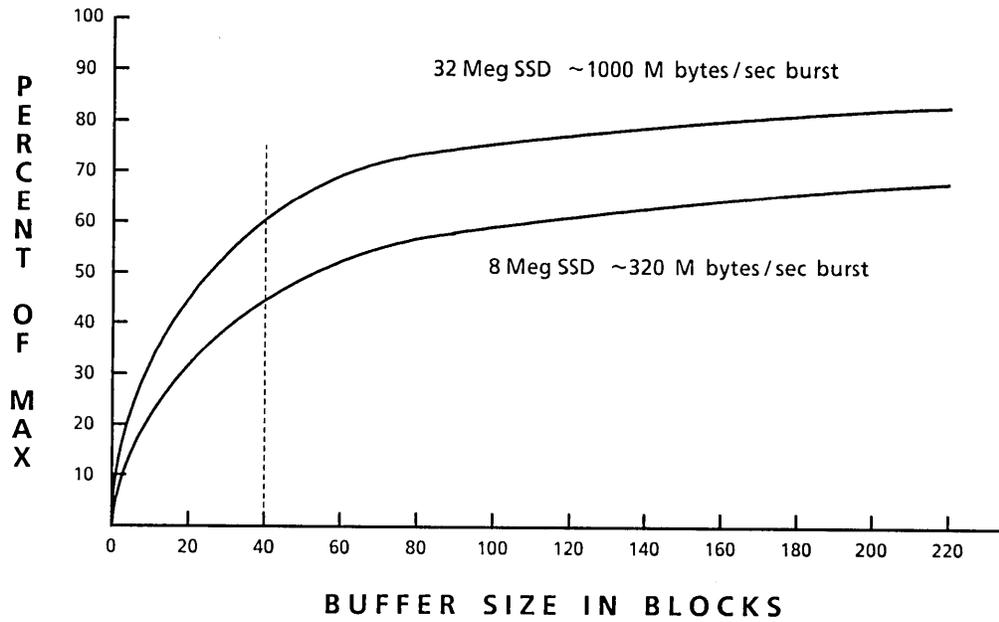
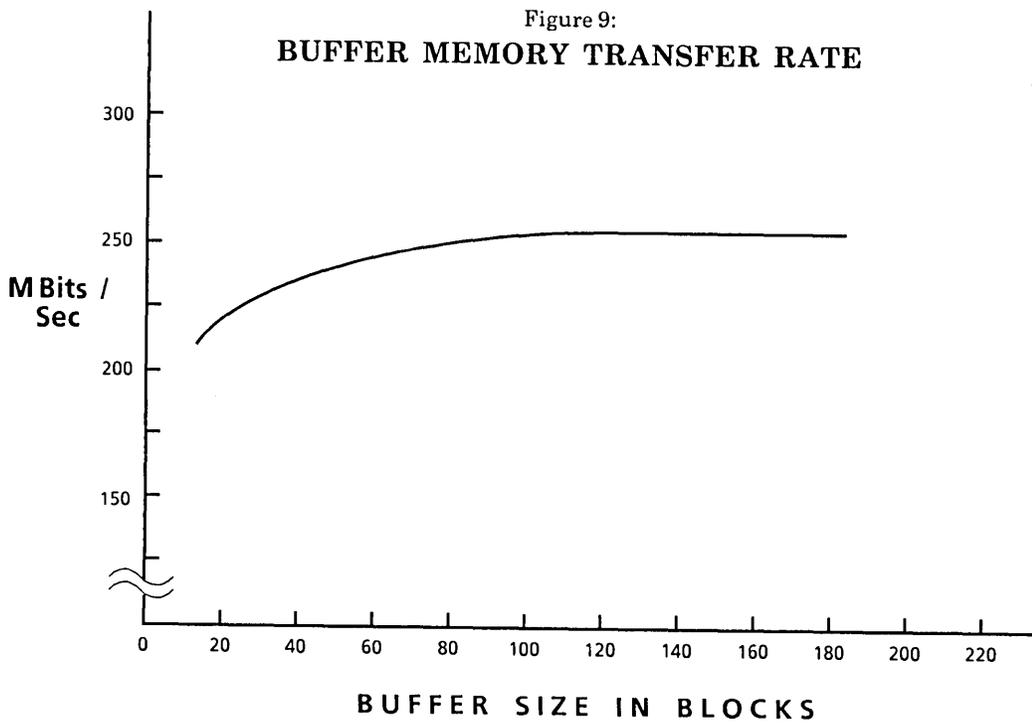
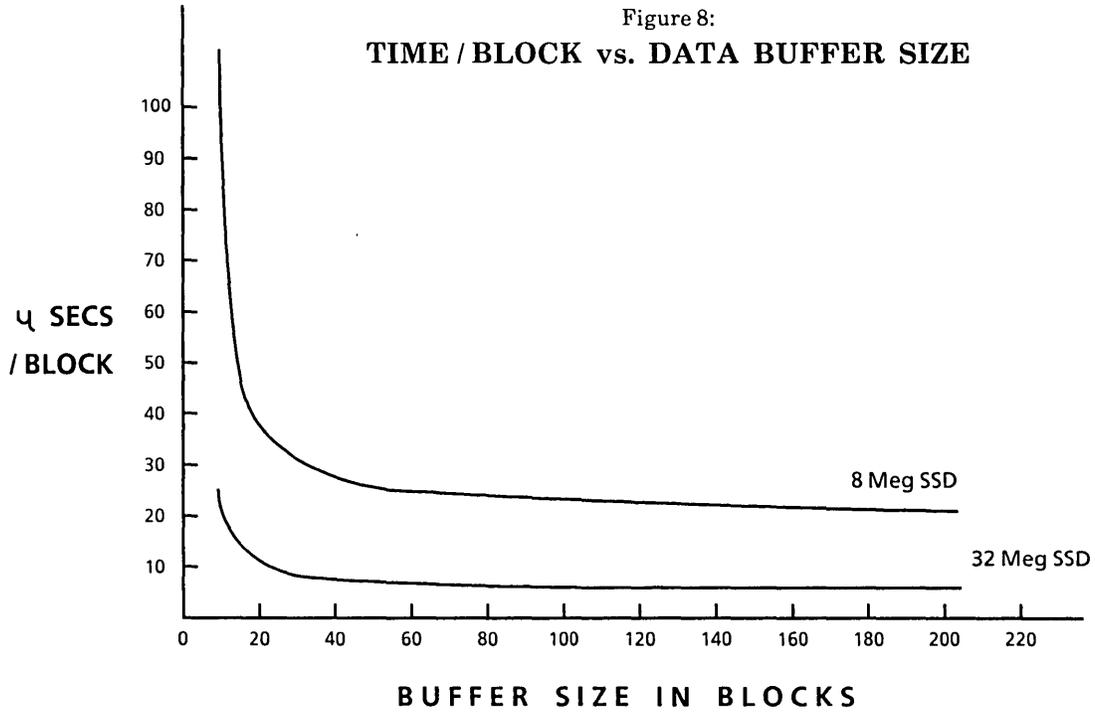


Figure 7:
SEQUENTIAL SSD TRANSFER RATE





SOFTWARE PAGING FOR LARGE MATRICES ON CRAY X-MP

U. Detert

Zentralinstitut für Angewandte Mathematik
Kernforschungsanlage Jülich GmbH
West Germany

Abstract: CRAY computers provide high computational speed but only limited main memory resources. This paper presents a fast and flexible method for the handling of large 2-dimensional matrices by a software paging mechanism using matrix segmentations with user-defined shape and size of blocks. Emphasis is put on performance analysis (CPU time, I/O requests, data transfer) depending on the matrix segmentation chosen. Performance data is given comparing several standard I/O methods with the software paging method.

Introduction

The following paper presents a software paging mechanism developed for the handling of large 2-dimensional matrices stored on secondary storage.

The paging mechanism allows for access to following matrix substructures:

- rows (or part of),
- columns (or part of),
- diagonals (or part of),
- matrix elements,
- blocks (i.e. rectangular submatrices).

The principle of operation of the software paging method is that of most customary paging systems: the data to be handled is segmented into pages (called blocks in the following) and stored in secondary memory; if certain items of the data are required for the computation they are loaded into main memory (demand paging). These pages form the "working set" in memory (called data buffer in the following), (figure 1).

A disadvantage with most customary paging systems when handling matrices is that normally the "natural" order of matrix elements in main memory is kept when the matrix is segmented into pages. In FORTRAN this means that pages comprise a few number of columns whereas rows are spread over a large number of pages.

The paging mechanism presented here differs in two ways from this. First, a block segmentation of the matrix is used; i.e. pages are submatrices of the given matrix. And second, blocks are not of fixed size but of user-defined size and shape. So, blocks (pages) may be square or rectangular submatrices. They may, however, also be a combination of one or more rows or columns if desired (called "horizontal" or "vertical" blocks, respectively).¹⁾

This concept provides high flexibility for the handling of matrices with different size and shape. Besides, best performance can be achieved by properly adopting the matrix segmentation to the underlying application and the resulting access pattern.

Performance

In the following, some performance considerations shall be discussed.

The first question of interest is the choice of the I/O routines used for the data transfer between data buffer and secondary storage. As all buffering is done in the data buffer of the paging system and the selection of the data items to be read or written is managed by the paging system, these I/O routines should not involve significant additional overhead for buffering the data in system I/O buffers. Secondly, the I/O routines should be well suited for direct or random I/O, as references to matrix blocks will in most cases be non-sequential.

The following CRAY I/O methods were investigated for use with the paging system:

- standard direct access I/O,
- READMS/WRITMS routines (random I/O, record addressable),
- READDR/WRITDR routines (unblocked random I/O, record addressable),
- PUTWA/GETWA routines (random I/O, word addressable),
- BUFFER IN/OUT (with SETPOS, GETPOS) (asynchronous direct access I/O).

1) A detailed description of the software paging method is given in:
U. Detert, Untersuchungen zur Implementierung eines Software-Paging-
Systems auf der CRAY X-MP.
Interner Bericht - KFA/ZAM 1/1985

Related to the criterions "CPU time", "number of I/O requests" and "amount of data transferred" for non-sequential data references READDR/WRTTDR on the one hand and BUFFER IN/BUFFER OUT on the other hand proved to be comparably qualified whereas all other routines exhibited either high CPU time consumption or a bad utilization of the I/O buffers. As READDR/WRTTDR is especially well suited for long records and requires no system I/O buffers, these routines were selected for use with the paging system.

The overall performance of the paging system is much influenced by the proper choice of the size and shape of the matrix blocks and the number of blocks kept in main memory at a time. For mere access to rows it is obviously optimal to choose "horizontal" blocks containing one or more rows each, at least one of them being in main memory at a time (figure 2). Correspondingly "vertical" blocks are optimal for the access to columns. If the dimensions of the matrix are $n \times m$ and the dimensions of the blocks are $s \times z$, then there are $1/s$ I/O operations to be performed per row of the matrix for a segmentation with horizontal blocks, provided that all rows of the block can be used before the block is overwritten.

The vector length for copying out one row from the data buffer to the user area is z , the length of the complete row, because each block contains complete rows. The amount of data transferred for access to each row is z (again under the assumption that the whole block can be used before it is overwritten) so there is no overhead in the amount of data transferred.

If access is to rows and columns with equal frequency, horizontal or vertical blocks are not reasonable, as they lead to an enormous amount of I/O operations and data transfer. In this case it can be proved that rectangular blocks are optimal with as many blocks per row as there are blocks per column. For square matrices this means that square blocks should be used. In this case $k_1 := n/s$ is the number of blocks per column and $k_2 := m/z$ the number of blocks per row (and normally $k_1 = k_2$). As one block is common to rows and columns, the data buffer should contain at least $k_1 + k_2 - 1$ blocks in order to enable the referencing of adjacent rows and columns without additional I/O operations (figure 3). In this case $m/(z \cdot s)$ I/O operations per row and $n/(s \cdot z)$ I/O operations per column are required. The amount of data transferred per row is m and the data transferred per column is n , if the whole data buffer can be utilized before blocks are overwritten. The vector length for copying out the data is s in the case of columns and z in the case of rows, which is a significantly smaller vector length than in the case of horizontal blocks or vertical blocks. So for the sake of good I/O and CPU performance blocks should not be too small.

Figure 4 shows the effect of reducing the block size constantly from 240 x 240 (which is the whole matrix) to 8 x 8 for the example of the LINPACK program SGEFA/SGESL for the solution of linear systems. For blocks significantly smaller than 40 x 40, CPU time and the number of I/O requests increase dramatically.

For the same program, examinations were made to find out the optimal number of blocks to be preserved in the data buffer. Figure 5 shows CPU time and the number of I/O requests for a fixed block size of 20 x 20 and a variable number of blocks in the data buffer ranging from 144 to 6 blocks. As access is 50 times more often to columns than to rows in this program, there is no actual need to buffer rows and columns (which would require a minimum number of 23 blocks in main memory). If however less than 12 blocks are kept in data buffer (which is the number of blocks required for the access to one complete row or column) "page flatterring" occurs, resulting in an enormous amount of CPU time and I/O requests.

Taking into consideration the much more frequent access to columns than to rows, square blocks possibly might not be the best choice in this case. Figure 6 shows CPU time, number of I/O requests, and number of disk sectors moved for the same program and various matrix segmentations (NBK is the number of blocks kept in data buffer).

For all runs shown in figure 6 the amount of memory reserved for the data buffer was about the same, however, in runs 1 to 3 (block sizes 20 x 20, 30 x 10 and 10 x 30) blocks for the access to rows and columns were kept in memory (i.e. $NBK = k_1 + k_2 - 1$), whereas in runs 4 and 5 (block sizes 240 x 40 and 80 x 40) only blocks for the access to columns were buffered ($NBK = k_1$). In the first case (buffering of rows and columns) square blocks are best; both, run 2 and run 3 show higher CPU time and I/O demands than run 1 due to the use of non-square blocks. Buffering of only columns with block size 240 x 40 and $NBK = 1$ (run 4) leads to an optimal behaviour concerning CPU time and number of I/O requests, as the very frequent access to columns is optimally realized in this case. However, the number of disk sectors moved goes up by a factor of 6 compared with run 1. This is due to the fact that every reference to a row forces the whole matrix to be read. Run 5 is a compromise between run 1 and run 4. Splitting up the "vertical" blocks of run 4 into three parts of size 80 x 40 each results in significantly less data transfer for row access and only negligible increase in CPU time and number of I/O requests. Figure 7 recalls the measurements for this example.

A performance comparison between the software paging system and some "standard" I/O methods is given in figures 8 and 9. Figure 8 shows performance data for a matrix multiply with matrix size 500 x 500 carried out in a row by row fashion. For this simple example access to rows of the matrix is completely sequential. So, matrices can be stored on secondary storage each row being one logical record and sequential and direct access I/O routines can be used to handle them. For the software paging system a block segmentation with horizontal blocks was used selecting block sizes with nearly optimal behavior (restrictions had to be accepted to meet buffer size requirements). Concerning CPU time the software paging system beats all I/O routines except BUFFER IN/BUFFER OUT which is slightly faster. With regard to the number of I/O requests, however, standard sequential I/O and BUFFER IN/BUFFER OUT are significantly better. Indeed, it is very difficult to beat sequential I/O by any means of direct access I/O.

The second example (figure 9) is a simple matrix traversal where the matrix is written row by row in forward direction and is read backwards. The whole procedure is repeated ten times. Here, the software paging system is best with regard to CPU time and the number of I/O requests. Concerning the amount of disk sectors moved, only READDR/WRITDR performs better than the paging system. This is due to the fact that with READDR/WRITDR each logical record corresponds to one physical record. Hence, no read is necessary before a write operation can be executed (a disadvantage of READDR/WRITDR, however, is the very large number of I/O requests). All other routines exhibit a very unsatisfactory utilization of the I/O buffers. Especially standard direct access I/O routines show poor performance in this respect.

A final assessment of the paging performance is given in figure 10. For various matrix sizes ranging from 50 x 50 to 1000 x 1000 CPU time and the number of I/O requests are given for the above mentioned LINPACK program. For comparison, both the CPU time with use of the paging system and without its use together with the ratio of both are represented. For a fixed upper limit of about 11000 words for the data buffer this ratio is almost a constant. For a 1000 x 1000 matrix an additional run with an increased buffer size of about 100,000 words (i.e. 10 % of the matrix are kept in main memory) shows that CPU time and the number of I/O requests can further be reduced.

Conclusion

The software paging method presented is a means designed for the fast and flexible handling of large 2-dimensional matrices not fitting into main memory.

The concept of user-defined matrix segmentations gives the ability to adopt the paging system to various applications with different I/O demands.

A comparison of the paging system with standard I/O methods shows satisfactory performance of the software paging system even in those simple cases where standard I/O methods can be applied. The applications aimed at with the paging system, however, lie far beyond this.

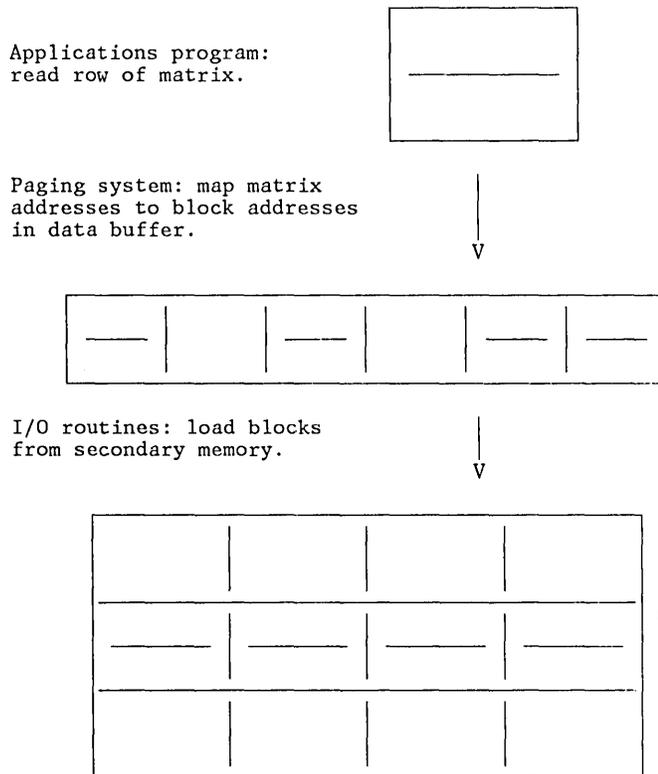


Figure 1. Virtual memory concept

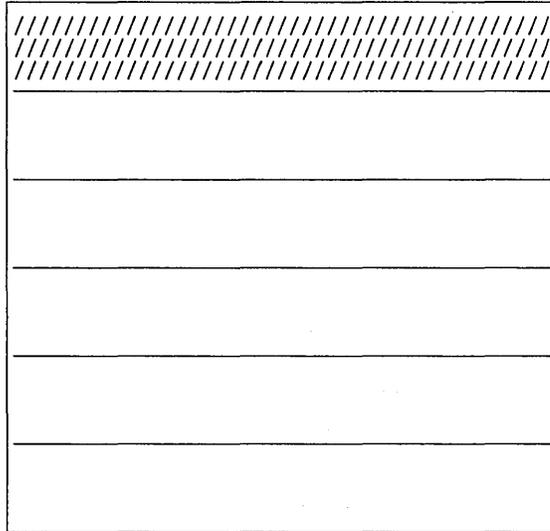


Figure 2. Matrix segmentation with "horizontal" blocks.

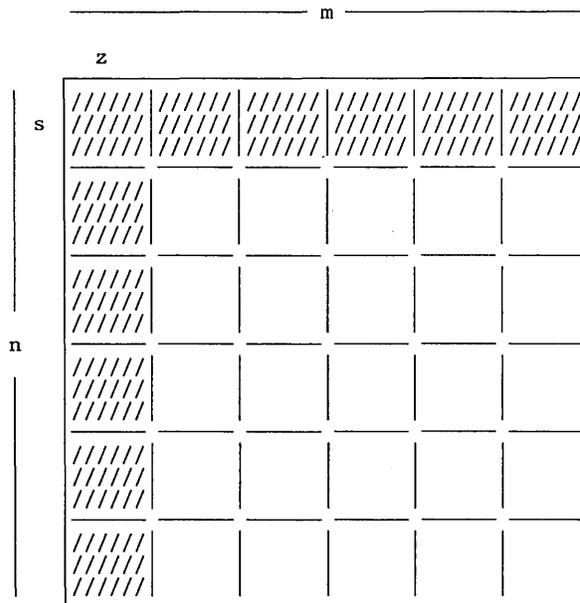


Figure 3. Matrix segmentation with square blocks.

Paging Performance for Variable Block Size

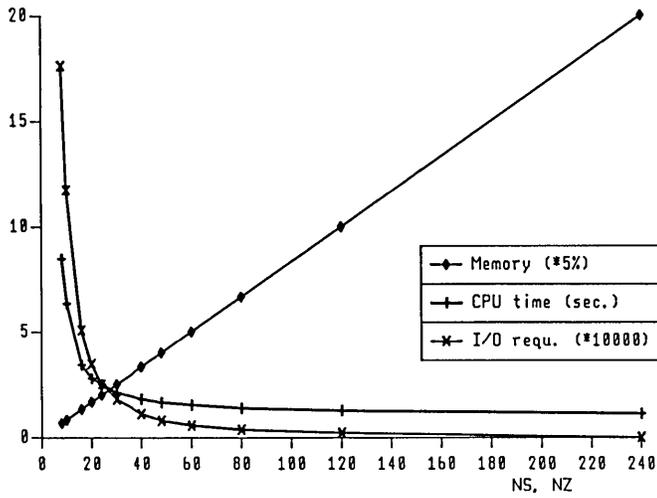


Figure 4. Paging performance depending on block size.

Paging Performance for Variable Number of Blocks in Main Memory

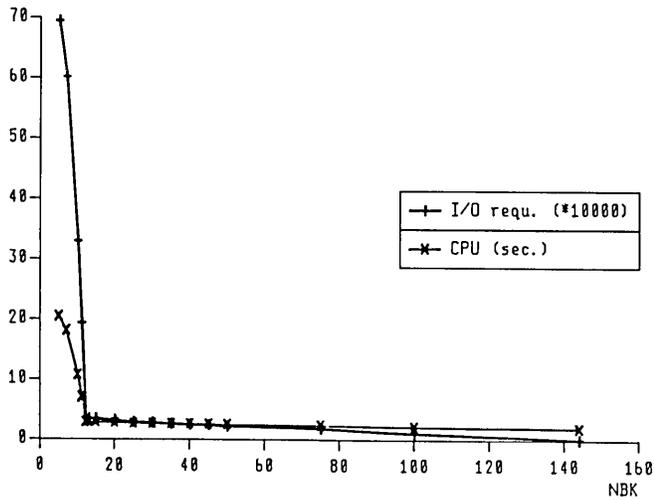


Figure 5. Paging performance depending on the number of blocks in data buffer.

Performance for Different Matrix Segmentations

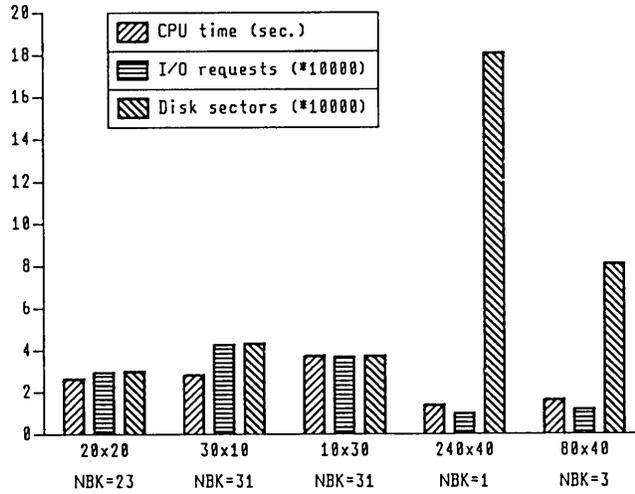


Figure 6. Paging performance for various matrix segmentations.

No.	s	z	NBK	Buffer size	CPU	I/O requ.	Disk sect.	I/O wait
1	20	20	23	9200	2.636	29256	29729	15.7
2	30	10	31	9300	2.819	42615	43090	19.0
3	10	30	31	9300	3.743	36775	37248	17.0
4	240	40	1	9600	1.372	9569	180486	23.0
5	80	40	3	9600	1.613	11534	80620	13.7

Figure 7. Measurements for figure 6.

I/O method	CPU time	I/O requ.	Disk sect.	I/O wait	Buffer size
sequential	12.73	4195	248828	26.2	16384
direct access	16.33	36705	254896	36.3	16384
BUFFER IN/OUT	6.64	3022	248828	13.8	16384
READMS/WRITMS	19.73	19617	247861	33.1	16384
READDR/WRITDR	9.60	253079	253408	1:21	0
software paging	6.97	11512	255424	33.5	16443
without I/O	4.14	-	-	-	750000

Figure 8. Performance data for matrix multiply with different I/O methods.

I/O method	CPU time	I/O requ.	Disk sect.	I/O wait	Buffer size
sequential	0.69	5034	43943	7.8	4096
direct access	0.63	15503	54547	12.3	4096
BUFFER IN/OUT	0.43	5004	43941	9.4	4096
READMS/WRITMS	0.91	9540	19085	6.5	4096
READDR/WRITDR	0.25	10070	10397	5.1	0
software paging	0.25	2017	15910	4.4	4177

Figure 9. Measurements for matrix traversal with different I/O methods.

Rank	Block size	Buffer size	I/O requ.	CPU with	CPU without	Ratio CPU
50	10x10	613	1762	0.12	0.013	9.0
100	50x25	2570	2800	0.28	0.050	5.6
150	50x30	4587	5976	0.64	0.12	5.3
200	50x40	8100	8216	1.14	0.24	4.9
250	64x32	8319	17410	1.88	0.39	4.8
300	100x30	9117	23284	2.83	0.61	4.7
600	200x15	9297	182765	15.10	3.16	4.8
1000	200x10	11063	889862	61.21	12.0	5.1
1000	200x100	100163	92233	40.74	12.0	3.4

Figure 10. Performance of LINPACK program SGEFA/SGESL with and without software paging.

NEW COS FEATURES

Clay Kirkland

Cray Research, Inc.
Mendota Heights, MN

1. AQIO.
2. ASSIGN CARD EXTENSIONS.
3. BACKDOOR TO SSD.
4. CIRCULAR I/O IMPROVEMENTS.

A Q I O.

1. ASYNCHRONOUS QUEUED INPUT OUTPUT.
2. SSD TRANSFERS ONLY ON COS 1.14.
3. DISK TRANSFERS ON COS 1.15.
4. FILES MUST BE PREALLOCATED.

A Q I O. ASYNCHRONOUS QUEUED INPUT/OUTPUT.

1. USER NOW HAS THE ABILITY TO QUEUE UP MULTIPLE INPUT/OUTPUT REQUESTS TO COS.
2. REQUESTS CAN BE DYNAMICALLY ADDED TO THE QUEUE BY THE USER AND MAY CONTAIN INTERMIXED READ AND WRITE REQUESTS.
3. REQUESTS PACKETS CONTAIN USER MEMORY ADDRESS, DISK ADDRESS, NUMBER OF DISK BLOCKS INVOLVED IN THE TRANSFER, AND A TRANSFER DIRECTION.
4. USER ALSO CAN INITIATE COMPOUND AQIO REQUESTS TO COS. THE COMPOUND REQUEST IS THE SAME AS THE REGULAR REQUEST WITH THE ADDITION OF A SKIP INCREMENT ON DISK, A SKIP INCREMENT IN MEMORY, AND AN INCREMENT COUNT. THIS CAN DRAMATICALLY LOWER SYSTEM OVERHEAD BY PASSING AN I/O DO LOOP TO THE LOWEST LEVEL OF THE OPERATING SYSTEM.

A Q I O.

F O R T R A N C A L L A B L E I N T E R F A C E .

1. ASSIGN FILE TO AND PREALLOCATE IT.
2. OPEN THE FILE BY CALLING AQOPEN.
CALL AQOPEN(AQP(1),(IREQ*8)+32,'FT01'L,0,ISTAT)
3. WRITE ON THE FILE.
CALL AQWRITE(AQP(1),ARRAY,BLOCK,NUMBLKS,ID,FIRE,IS)
4. READ THE FILE.
CALL AQREAD(AQP(1),ARRAY,BLOCK,NUMBLKS,ID,FIRE,IS)
5. COMPOUND WRITE ON THE FILE.
2 CALL AQWRITEC(AQP(1),ARRAY,MEMSTRD,BLOCK,
NUMBLKS,DSKSTRD,INCS-1,ID,FIRE,IS)
6. COMPOUND READ ON THE FILE.
2 CALL AQREADC(AQP(1),ARRAY,MEMSTRD,BLOCK,
NUMBLKS,DSKSTRD,INCS-1,ID,FIRE,IS)
7. CHECK STATUS ON THE FILE.
CALL AQSTAT(AQP(1),REPLY,REQID,STATUS)
8. WAIT UNTIL I/O QUIET ON FILE
CALL AQWAIT(AQP(1),STATUS)
9. CLOSE FILE.
CALL AQCLOSE(AQP(1),STATUS)

PERFORMANCE RESULTS OF AQIO ON SN 201 TO SSD.

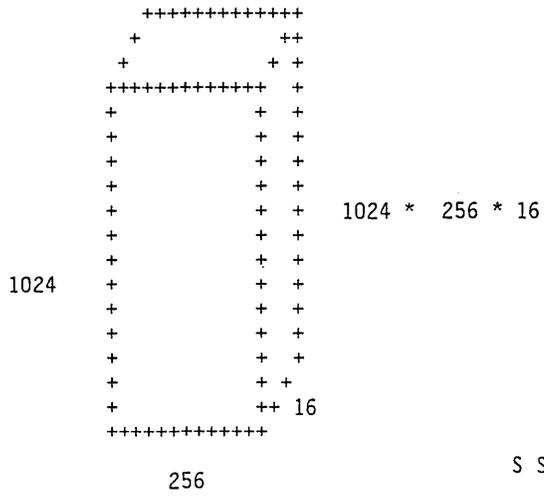
SIZE OF TRANSFER IN WORDS	RATE IN MEGAWORDS/SEC.			
1024 NORMAL READ/WRITE.	2.50	"	3.34	"
1024 AQIO 20 REQUESTS.	12.51	"	12.72	"
1024 AQIO COMPOUND REQ OF 20.	26.15	"	26.32	"
2048 NORMAL READ/WRITE.	4.89	"	6.47	"
2048 AQIO 20 REQUESTS.	24.95	"	25.37	"
2048 AQIO COMPOUND REQ OF 20.	52.08	"	52.35	"
4096 NORMAL READ/WRITE.	10.01	"	13.35	"
4096 AQIO 20 REQUESTS.	49.48	"	50.11	"
4096 AQIO COMPOUND REQ OF 20.	98.01	"	100.76	"

TO ACHIEVE THE SAME TRANSFER RATE AS THE COMPOUND REQUEST OF 1024 WORDS USING NORMAL READ/WRITE, WE NEED A TRANSFER SIZE OF 10752 WORDS. TO DUPLICATE THE 2048 COMPOUND RATE WE NEED A TRANSFER SIZE OF 43008 WORDS, AND TO DUPLICATE THE 4096 COMPOUND RATE WE NEED TO MOVE 82944 WORDS.

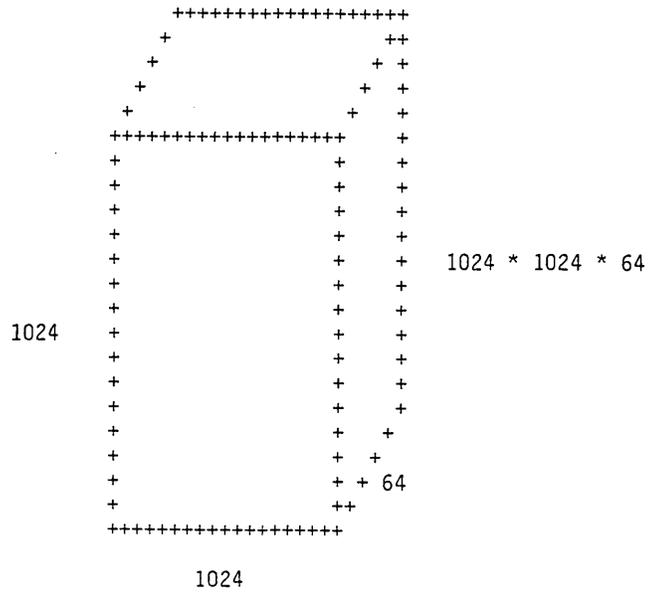
W H Y D I S K A Q I O ?

1. TIME TO DO F\$RDC OR F\$WDC CALL. (NORMAL READ/WRITE)
300 - 400 MICROSECONDS.
2. TIME TO DO F\$QIO CALL.
400 - 1000 MICROSECONDS.
3. TIME TO ADD REQUEST TO THE QUEUE.
5 - 6 MICROSECONDS.
4. THUS TO DO 100 WRITE REQUESTS ON A DATASET.
100 * 350 = 35000 MICROSECONDS (F\$WDC)
600 + 5 * 100 = 1100 MICROSECONDS (F\$QIO)
5. PAYOFF IN REDUCED SYSTEM OVERHEAD.
6. USER - LIB - EXEC - EXP(CIO) - DQM - CIO -
EXEC - LIB - USER.

CENTRAL MEMORY CUBE.



S S D CUBE.



A Q I O : E X A M P L E .

```
JOB,JN=TAPE,T=7,SSD=131080.
ACCOUNT,-----
ASSIGN,DN=FT01,DV=SSD-0-20,LIM=131080.
WRITEDS(DN=FT01,NR=131,RL=512000.
CFT.
LDR.
/EOF

PROGRAM TEST
PARAMETER (IL = 1024),(JL = 256),(KL = 16)
COMMON /A/ A(IL,JL,KL),B(IL,JL,KL),AQP(1024)
INTEGER FIRE,BLOCKS,STATUS

C        SSD SIZE IS SSD(1024,1024,64)

CALL INIT    ! OPEN AQIO FILE AND INITIALIZE.
FIRE        = 0
BLOCKI      = 0
NBLOCKS     = 512
ID          = 0
DO 20 J = 1,8
BLOCK       = BLOCKI
DO 10 I = 1,4

C        READ IN 16 SUB PLANES FROM SSD WITH ONE CALL.

CALL AQREADC(AQP,A,(IL * JL),BLOCK,NBLOCKS,
+            2048,(KL-1),ID,FIRE,STATUS)
ID = ID + 1
CALL AQWAIT(AQP,STATUS)
CALL PROCESS
BLOCK = BLOCK + NBLOCKS
10        CONTINUE
BLOCKI = BLOCKI + (2 * 256 * 16)
20        CONTINUE
STOP
END
/EOF
```

ASSIGN CARD EXTENSIONS.

1. ASSIGN, DN=FT01, , , , SZ=180, INC=180, C.

SZ = DATASET SIZE IN DECIMAL SECTORS. IF BOTH SZ AND INC ARE SPECIFIED, SZ IS USED INITIALLY AND INC IS USED ON SUBSEQUENT ASSIGNS.

INC = NUMBER OF DECIMAL SECTORS TO ALLOCATE EACH TIME ALLOCATION OCCURS.

C = CONTIGUOUS SPACE ALLOCATION. ALLOCATE CONTIGUOUS SPACE EQUAL TO THE SZ PARAM OR THE INC PARAM OR THE SYSTEM DEFAULT. IF C IS NOT SPECIFIED, THE SYSTEM TRIES TO FIND CONTIGUOUS SPACE ON THE SELECTED DEVICE ONLY. IF C IS SPECIFIED AND NOF IS NOT THE SYSTEM SEARCHES ON EVERY AVAILABLE DEVICE.

2. ASSIGN, DN=FT01, , , , DT=DT1:DT2:DT3:DT4.

DT = DT1:DT2:DT3:DT4 DEVICE TYPE. DT1 THROUGH DT4 ARE DEVICE TYPES IN THE SYSTEM. IF LDV WAS NOT SPECIFIED, AN ATTEMPT WILL BE MADE TO ALLOCATE SPACE FIRST ON DT1. OVERFLOW WILL GO TO DT2 TO DT3 AND DT4. IF THE SPACE REQUIREMENT CANNOT BE MET ON DEVICE TYPES AND NOF IS NOT DECLARED, OVERFLOW WILL CONTINUE ON THE DEFAULT DEVICES. IF AN LDV WAS NAMED AND SPACE WAS NOT AVAILABLE ON THAT LDV, THEN DEVICE SELECTION GOES ACCORDING TO DT. ALLOWABLE DT'S ARE COS SUPPORTED DEVICES DD19, DD29, DD39, DD49, SSD AND EBM.

3. ASSIGN, DN=FT01, , , , ST=SCR.

ASSIGN, DN=FT02, , , , ST=PERM.

T = STORAGE TYPE. DEFAULT WILL BE INSTALLATION PARAMETER. CAN BE SCR (SCRATCH) OR PERM (PERMANENT)..

ENHANCEMENTS FOR DEVICE SELECTION.

1. THE CURRENT DEFAULT DEVICE SELECTION IN DQM IS ROUND ROBIN ON THE DEFAULT DEVICES. THIS METHOD IS SUFFICIENT IF ALL THE DEVICES ARE THE SAME TYPE AND SPEED, BUT WE NOW HAVE DEVICES THAT VARY WIDELY IN SIZE, SPEED AND NUMBER OF CHANNELS.
2. THE SELECTION METHOD WILL BE CHANGED TO SELECT A DEVICE ON THE CHANNEL THAT IS LEAST ACTIVE. THIS SHOULD CAUSE THE CHANNEL ACTIVITY ON ALL CHANNELS TO BE MORE EVEN AND REDUCE THE I/O WAIT TIME IN THE SYSTEM.
3. COS WILL ALSO ALLOW A DEVICE TO BE SCRATCH WHETHER IT BE A PRIVATE (REQUEST BY NAME) OR PUBLIC. IT WILL ALSO BE POSSIBLE FOR A SITE TO DECLARE DEFAULT SPACE AS SCRATCH OR PERMANENT.
4. IF DEFAULT SPACE IS SCRATCH AND SSD IS SET TO A SCRATCH DEVICE, THEN THE SSD SHOULD GET MOST OF THE SMALL SCRATCH FILES BECAUSE THE DEVICE SELECTION WILL FAVOUR IT FAST CHANNEL.

SSD BACKDOOR.

1. USER NOW HAS ABILITY TO ISSUE SYSTEM FUNCTION TO COPY PORTIONS OF UNBLOCKED FILES FROM/TO SSD (OR BUFFER MEMORY) TO/FROM DISK.
2. USER NEEDS TWO DSP'S, ONE FOR INPUT FILE AND ONE FOR THE OUTPUT FILE.
3. USER PUTS STARTING AND ENDING BLOCK NUMBER IN A SPECIAL WORD OF THE INPUT DSP AND PUTS THE STARTING BLOCK NUMBER IN THE OUTPUT DSP.
4. USER THEN SETS S1 = INPUT DSP ADDRESS, S2 = OUTPUT DSP ADDRESS AND ISSUES A F\$CPY CALL.
5. I/O IS DONE ASYNCHRONOUSLY BOTH DSP'S ARE BUSY AND STATUS IS DONE IN THE NORMAL WAY.
6. PACKET TO THE IOP HAS BITS SET IN IT TO INDICATE THE TARGET MEMORY FOR THE FILE TRANSFER (CPU MEMORY, SSD MEMORY OR BUFFER MEMORY).
7. BUFFER MEMORY TRANSFERS REQUIRE NO NEW HARDWARE
8. SSD MEMORY TRANSFERS REQUIRE 1 HIGH SPEED (850 MEGABITS/SEC TYPICALLY ON THE XIOP) CHANNEL AND SOME SPECIAL MODULES IN THE SSD.

CIRCULAR I/O IMPROVEMENTS.

1. MORE CHANGES TO THE ASSIGN CARD.
ASSIGN, DN=FT01, XSZ=MAX:MIN.
2. XSZ IS USED TO SET THE MAXIMUM AND MINIMUM TRANSFERS THAT COS WILL DO ON THIS FILE.
3. IN THE OLD DAYS THERE WAS NO MINIMUM, MAXIMUM WAS HALF OF THE BUFFER.
4. NOW (COS 1.15) INITIAL SIZE IS MAXIMUM AND ON SUBSEQUENT CIRCULAR CHAINING COS WILL TRY TO USE MINIMUM.
5. TAPE FILES DO NOT USE THIS. MOST EFFECTIVE ON SSD OR OTHER FAST DEVICES.
6. USER COULD FOR INSTANCE SET UP A 12 SECTOR BUFFER AND SET THE MINIMUM TRANSFER TO 3 SECTORS. THERE WOULD THEN BE 4 TRANSFERS TO FILL THE BUFFER AND THE USER WOULD BE RECONNECTED AFTER THE FIRST 3 SECTORS SO THE ABILITY TO KEEP THE STREAM GOING IS ENHANCED.
7. F\$RCL IS CHANGED. USER IS RECONNECTED MUCH FASTER.

DD-39 STREAMING TESTS THROUGH A SINGLE IOP

RECORD SIZE = 2 CYLINDERS.

STREAMS	WRITES		READS	
	MW/SEC	MB/SEC	MW/SEC	MB/SEC
1	.729	5.83	.728	5.82
2	.731	5.84	.728	5.82
3	.730	5.84	.726	5.81
4	.731	5.85	.728	5.83
5	.729	5.83	.727	5.81
6	.728	5.82	.722	5.78
7	.724	5.79	.715	5.72
8	.689	5.51	.676	5.41

DD-49 STREAMING TESTS THROUGH A SINGLE IOP

RECORD SIZE = 2 CYLINDERS.

STREAMS	WRITES		READS	
	MW/SEC	MB/SEC	MW/SEC	MB/SEC
1	1.21	9.66	1.21	9.65
2	1.21	9.66	1.21	9.64
3	1.21	9.67	1.21	9.65
4	1.21	9.66	1.21	9.67
5	1.20	9.57	1.13	9.09
6	1.10	8.82	.97	7.76
7	1.07	8.57	.55	4.39
8	1.07	8.55 *	.50	4.01

* TOTAL RATE OF 547 MEGABITS / SEC OVER CHANNEL

B M R T R A N S F E R S P E E D S.

RECORD SIZE IN WORDS	ACCESS TIME MICROS	WRITE RATE MWORDS/SEC.	READ RATE MWORDS/SEC.
512	995/1157	.51	.44
1024	1105/1056	.92	.97
2048	1636/1309	1.25	1.57
4096		2.43	2.49
8192		2.97	3.30
10240		3.36	3.54
43008		4.35	4.34
65536		4.73	4.70
102400		4.81	4.50

S S D T R A N S F E R S P E E D S .

RECORD SIZE IN WORDS	ACCESS TIME MICROS	WRITE RATE MWORDS/SEC.	READ RATE MWORDS/SEC.
512	408/306	1.25	1.67
1024	408/306	2.50	3.34
2048	418/316	4.89	6.47
4096		10.01	13.35
8192		19.53	21.81
16896		34.48	24.39
32768		64.00	45.37
65536		79.40	89.50
102400		112.59	119.04

T A P E S :

1. TRANSPARENT OR INTERCHANGE FORMATS
2. IMPLICIT OR EXPLICIT CONVERSION.
3. CDC, IBM, SUPPORTED ON COS 1.14.
4. VMS SUPPORTED ON COS 1.15.
4. 3480 SUPPORT ON COS 1.15.

T A P E S : C H A R A C T E R C O N V .

```
JOB,JN=TAPE,T=7,*6250=1.
ACCOUNT,-----
ACCESS, DN=FT01,DF=IC,DT=*6250,MBS=4000,CV=ON,
CS=EB,FD=IBM,RF=FB,RS=80.
```

CFT.

LDR.

/EOF

```
PROGRAM TEST
DIMENSION IA(10)
```

```
DO 10 I = 1,1000
READ(1,100,END=200) IA
FORMAT(10A8)
100 CONTINUE
```

```
200 CONTINUE
STOP
END
```

/EOF

TAPES : CHARACTER CONV.

JOB,JN=TAPE,T=7,*6250=1.
 ACCOUNT,-----
 ACCESS,DN=FT01,DF=IC,DT=*6250,MBS=4000,CV=ON,
 CS=EB,FD=IBM,RF=FB,RS=80.

CFT.

 LDR.

/EOF

```

PROGRAM TEST
DIMENSION IA(10)

DO 10 I = 1,1000
READ(1,100,END=200) IA
FORMAT(10A8)
CONTINUE

100
10
CONTINUE

200
CONTINUE
STOP
END
  
```

/EOF

TAPES : FLOATING PT. CONV.

JOB,JN=TAPE,T=7,*6250=1.
 ACCOUNT,-----
 ACCESS,DN=FT01,DF=IC,DT=*6250,MBS=4000,CV=ON,
 CS=EB,FD=IBM,RF=F,RS=4000.

CFT.

 LDR.

/EOF

```

PROGRAM TEST
DIMENSION A(1000)

DO 10 I = 1,1000
READ(1,END=200) A
CONTINUE

10
CONTINUE

200
CONTINUE
STOP
END
  
```

/EOF

TAPES : EXPLICIT CONVERSION.

JOB,JN=TAPE,T=7,*6250=1.
 ACCOUNT,-----
 ACCESS,DN=FT01,DF=IC,DT=*6250,MBS=32760.

 CFT.

 LDR.

/EOF

```

PROGRAM TEST
DIMENSION A(10000),B(10000)

EOFST = 0.0
DO 10 I = 1,1000
BUFFER IN(1,1)(A(1),A(10000))
IF(UNIT(1).EQ.EOFST) GO TO 200
IL = ILENGTH(1)
CALL MYCONV(A,IL,B)
WRITE(2)(B(J),J=1,IL)

10
CONTINUE
  
```

```

200
CONTINUE
STOP
END
  
```

/EOF

CUSTOMER EXPERIENCE WITH DD-49 DISKS

Mostyn Lewis

Chevron Oil Field Research Company
La Habra, CA

Chevron Oil Field Research Company (COFRC) had a recent delivery of a Cray X-MP/48. The configuration includes 28 disks, 20 of which are the DD-49 variety (IBIS). Four DD-49's were delivered in February of 1985 and the remainder at the beginning of May 1985. The original four disks had one incident and eight of the remainder suffered incidents. Some trouble occurred because 19 spindle motors in 19 drives were changed due to a lack of thermal protection on the original motors. A head failed on one drive apparently due to a "noisier" motor. The cure was to install ferrite "beads" on a lead between the servo and HDA (Head Disk Assembly). Only half of our disks had "beads" and we wondered why. Five power supplies were changed -- caused by a bad part, a bridge rectifier, in four cases, and by faulty quality assurance in the other (loose screws!). The power supply problems caused fallout resulting in bad modules and crowbars (circuit breakers). Six HDA's failed and were replaced. At the time of the failures, we believed they were due to bad electronics inside the HDA, head crashes and mis-reading of the servo track(s) and/or bad servo head. It transpired, also, that we had some incompatibilities between HDA's and supporting cards in the drive.

Various rumours arose concerning the unsuitability of travel for HDA's and bad packaging.

All in all, it may have seemed a gloomy picture. However, as an early customer, COFRC was prepared for teething problems and considered the above as examples. If you're a pioneer, you expect to forge the pathway for others. The disks gave a level of performance necessary to complement the X-MP/48 and it has been easy to achieve nine megabytes per second transfer rates.

We expect things to quiet down and for Cray and IBIS to achieve a stable hardware product in light of "post-mortem" evidence. The speaker is personally in favor of these "leading edge" disks and has confidence in the amelioration of the current situation.

Report on the Graphics Session

H. E. Kulsrud

IDA/CRD
Princeton, NJ

The graphics session began with a paper entitled "Digital Image Synthesis on the Cray" by Grey Lorig of CRI. Grey described the OASIS System for digital animation which is being developed at CRI. The goals for this system are an open environment, clean interfaces and portability. Grey is uniting some established packages and adding new ones. OASIS will handle interactive modeling, image rendering and ray tracing etc. This will probably not be a Cray supported package.

A Video film made by Zero One Systems for Nasa Ames was shown. Interactive Graphics on an Iris Workstation using calculations from a Cray was dramatically demonstrated. This film showed a real application, simulation of fluid flow over the space shuttle, and was a model of how effective such an equipment marriage can be.

John Aldag of CRI led a discussion on User Requirements for Graphics. John presented some issues discussed in a Strategic Planning paper he is preparing for Cray. His objective is to understand graphics requirements of Cray Users, raise awareness of CRI management, provide graphics performance in line with the computational power of a Cray and to enhance marketability. John presented two alternatives for Graphics, broad performance and high performance. Although the users seem to want both alternatives, the plea for high performance Graphics (50-100 megabytes or more) was stressed by the speakers. The question seems to be involved with whether CRI will release the high speed channel specifications to at least one graphics equipment company.

The concluding item of the session was the formal formation of the CUG special interest Committee on Graphics and Data Bases. The prior discussion revealed the need for such a Committee and its first meeting will take place on the opening Monday of the Seattle meeting.

ADDITIONAL REPORTS

PRESIDENT'S REPORT

M. G. Schomberg

AERE-Harwell
England

This is the sixteenth meeting of the Cray User Group and the meeting at which we will be taking the vote to incorporate CUG. I will be speaking about incorporation a little later.

I would like to remind you that your Board of Directors consists of:

Michael Schomberg	President
Laney Kulsrud	Vice President
Karen Friedman	Secretary
Bob Price	Treasurer
Jacqueline Goirand	Director at Large
Dave Lexton	Director at Large
Joe Thompson	Director at Large

The Monday prior to the start of this general meeting proved to be a very busy day during which many of the technical and administrative committees met. The Board of Directors and the Advisory Council also had meetings.

I am pleased to report that there are 69 sites who are paid-up Installation Members of the Cray User Group. In addition there are a few other sites who are represented here but who have not paid their fees. The Board of Directors will now be getting rather tougher with those sites who are not paying their fees and in future they will be excluded from meetings.

The funds of the Cray User Group stand at a very healthy \$9243 and these are deposited in an interest bearing account. Your Board of Directors have recommended that the annual fee for an Installation should remain at \$100 for the next year.

The Program Committee chaired by Laney Kulsrud and comprising the chairs of Special Interest Committees has again been very active as can be judged from the excellent programme which has been prepared for this meeting. The number of papers offered for CUG meetings continues to grow. Hence for the Seattle meeting in Spring of 1986 we have agreed to extend the Conference to three and a half

days. The meeting will therefore finish at mid-day on Friday. One of the objectives of increasing the length of the meeting is to reduce the number of parallel sessions.

In my report in the Spring I stated that your Board of Directors were concentrating their efforts onto two major objectives.

One of these objectives was to improve the technical communication between CUG and Cray Research Inc. Your Board sees this as working in two ways. First, user requests and requests from the Special Interest Committees are submitted to the User Requirement Committee which is chaired by Steve Niver. The User Requirement Committee examines each request in detail to ensure that it is substantive, technically viable and unambiguous. A list of about ten key items is then sent to Installation Delegates to cast their votes indicating the importance and relevance of each item to their installation. This has been accomplished this time and, following the approval of the Board of Directors these will be submitted as formal requests to Cray Research Inc. Steve Niver will be reporting fully on this later in the meeting.

The other way of improving technical communication is to establish a close technical working relationship between the Special Interest Committees and Cray Research Inc. technical staff. Cray staff should attend selected meetings of the Special Interest Committees and also Cray will submit some design documents to these committees for comment. These procedures are starting to work but not all the committees are yet fully active. In addition the interest from Cray has not yet been as positive as we would wish.

These problems are being addressed and I am confident that there is a genuine desire from all concerned to make this part of CUG as effective as possible.

The second major objective of your Board has been to incorporate the Cray User Group. Thanks to the efforts of Bob Price all the necessary preparatory work has now been completed. In a few moments I will be putting to you the motion:

"It is proposed by the Board of Directors of the Cray User Group to dissolve this Association. It is also proposed to transfer any and all assets, accounts receivable, membership, committee organization, and any established operations, plans or intentions of this Association to the Cray User Group, Incorporated."

Voting is to be by Installation Delegates only using the orange cards which have been issued.

Before taking the vote I would like to remind you of the main reasons for

Incorporation. These are:

- . to limit personal liability
- . to bind the officers of the organization against embezzlement
- . to facilitate insurance.

The structure, function and organization of the Cray User Group will not change as a result of incorporation.

Before closing my report I would like to offer my personal thanks to all the members of the Board of Directors and to the chairs of all the various committees for the tremendous amount of work they have put into the Cray User Group. We come have a long way over the last two years or so and this has only been possible by much hard work by a significant number of people. Thank you.

INCORPORATION OF THE CRAY USER GROUP

M. G. Schomberg

AERE-Harwell
England

The following motion was put to the general membership of the Cray User Group on 1 October 1985.

It is proposed by the Board of Directors of the Cray User Group to dissolve this Association. It is also proposed to transfer any and all assets, accounts receivable, membership, committee organization, and any established operations, plans or intentions of this Association to the Cray User Group, Incorporated.

A vote of Installation Delegates was taken on the above motion. The results were:

In favour - 29 votes

Against - none.

The motion was therefore carried unanimously and the Cray User Group became an Incorporated body on 1 October 1985.

Report of the Vice President

H. E. Kulsrud

IDA/CRD
Princeton, NJ

Preparation of the Program for Montreal proceeded without difficulties. There were no changes of personnel on the Program Committee and all the members were able to attend the Montreal meeting. With the more formal organization of the SICs there will now be committee members available for organizing sessions. However, responsibility for the parallel sessions will still lie with the SIC Chairs. The theme of multitasking and multiprocessing proved very timely and raised a number of questions of interest to the attendees. The theme and the excellence of the program contributed to the large attendance at the meeting. We were finally able to get a keynote speaker and in the future will attempt to coordinate the keynote and the meeting theme. As anticipated, overlap of material between sessions was the principal organization problem for this meeting. We finally changed the names of some of these sessions to reflect this overlap. Sixty-six people participated in the technical program.

Work has begun for the Seattle meeting. The theme is UNIX and several speakers have already been contacted. The Call for Papers for Spring 86 was included in the Stockholm proceedings and in the Montreal abstracts. The call will also be included in the first Seattle mailing.

Ray Benoit and his local arrangements committee did an excellent job - this is becoming a tradition at CUG. Faced with fifty more attendees than they had planned for, the committee was able to provide facilities for all. I am sure that the charm of Montreal and these arrangements also contributed to the attendance. Our sincere thanks to this committee.

Future CUG meetings are planned for:

May 5-9, 1986	Seattle, Washington	Boeing Computer Services
Sept. 29-Oct. 3, 1986	Garmisch-Partenkirchen, West Germany	DFVLR
Spring 1987	New York City, NY	Grumman Data Systems
Fall 1987	Bologna, Italy	CINECA
Spring 1988	Minneapolis, Minn.	University of Minn.

Report of the Program Committee

David Lexton

University of London Computer Centre
England

Seattle CUG

The program committee met at 5 p.m. on 2nd October 1985. Membership of the committee remains substantially unchanged.

Montreal Meeting

The local arrangements committee under Ray Benoit maintained the high standard for CUG organisation set in Stockholm. On this basis, the program committee was able to discuss some detailed criticisms of the program itself. First of all it was pointed out that the call for papers frequently does not reach the people in the organisation who would be interested in giving papers. Efforts would therefore be taken to make the call more eye-catching. There was agreement that the quality of papers in general sessions was good but that more time should be left for questions. In some cases, such as the front end session, it was felt that there had not been enough time allocated in the first place. Some presentations were thought to contain too much detail or not to be pitched at the right level. It was agreed that it would be desirable to have more presentations on applications. The view was expressed that CUG should be seeking from CRI statements of philosophy rather than project reports.

Conference Proceedings

The delay in getting papers in to the proceedings editor (Karen Friedman) had been the worst ever for the Stockholm Conference. Anyone not going to make the deadline was urged to contact the editor. It was agreed that, in future and where appropriate, contributors would be asked to bring their papers to CUG in a form ready to go into the proceedings.

The Seattle CUG will last three-and-a-half days, starting on Tuesday morning and ending at Friday lunchtime. It was agreed that up to three sessions in parallel would be allowed and rough equality maintained between general and parallel sessions. The theme of the meeting is UNIX and most of the suggestions for general sessions were related to the theme. These were COS-to-UNIX migration, UNICOS, multitasking, applications under UNIX, UNIX performance, UNIX security, SSD Management under UNIX, UNIX resource management, comparison of CX-OS and UNIX, Cray-2, UNIX and CTSS, managing very large resources under COS and UNIX, software reliability, Crayettes and, finally, at least one humorous talk.

The different committees requested the following numbers of sessions: Software tools, 3. Communications, 3. Operating Systems, 2. CTSS, 1. Performance, 2. Database and Graphics, 1. Operations, 3.

Garmisch-Partenkirchen CUG

It was agreed by the committee that the theme should be Applications and Algorithms. Reliability, Performance and Tuning would be the theme for the following meeting in New York.

ADDITIONAL INFORMATION

MONTREAL CRAY USER GROUP MEETING PARTICIPANTS BY ORGANIZATION

Organization	Representatives	Phone Numbers
Aramco Dhahran P.O. Box 5000 S. Arabia	Chuck Deprycker	96638766146
Atlantic-Richfield Oil & Gas 2300 Plano Parkway Plano, TX 75075	B.Y. Chin Chuck Murphy Dean Smith	(214) 422-6627 (214) 754-6612 (214) 754-6415
AT&T Bell Laboratories 600 Mountain Avenue Murray Hill, NJ 07974	Tony Shober	
Boeing Computer Services 565 Andover Park West 9C-01 Tukwila, WA 98188	David S. Dodson Kenneth W. Neves	(206) 575-5107 (206) 575-5074
Boeing Computer Services P.O. Box 24346 Seattle, WA 98124	Conrad Kimball Steve Niver Howard Schmeising	(206) 763-6410 (206) 763-5073 (206) 763-5069
CCVR Ecole Polytechnique 91128 Palaiseau CEDEX France	Serge Hardoin	6 941 82 00
CEA - France Centre de LIMEIL B.P. 27 94190 Villeneuve St. Georges France	Jacques David Martine Gigandet	4595 6289 4595 6184
CEA-CEV Unite de Calcul BP 7 77181 Courtry France	Joseph Harrar	1 868 8688
Centre de Caclul Vectoriel pour la Recherche Ecole Polytechnique 91128 Palaiseau CEDEX France	Maurice Benoit	6 941 82 00
Chevron Geosciences 2811 Hayes Road Houston, TX 77242	A.R. Bertrand David L. Miller	(713) 596-2515 (713) 596-2515

Chevron Oil Field Research Company 3282 Beach Blvd. La Habra, CA 90631	Annabella Deck Mostyn Lewis	(213) 694-9218 (213) 694-9235
CIFRAM CEN-Saclay BP 24 91190 Gif-sur-Yvette France	Jacqueline Goirand Regis Schoonheere	(6) 908 3841 (6) 908 6319
CINECA 6/3 Magnanelli Casalecchio di Reno 40033 Bologna, Italy	Marco Lanzarini Elda Rossi Bassini Sanzio	39 51 576541 39 51 576541 39 51 576541
Compagnie General de Geophysique 1 Rue Leon Migaux 91301 Massy France	Yves Gouedranche Claude Guerin	331 69 20 8408 331 69 20 8408
Cray Canada, Inc. 207 Place Frontenac Pointe Claire, Quebec H9R YZ7 Canada	Rejean Chartier	(514) 695-0210
Cray Canada, Inc. 4141 Yonge Street Toronto, Ontario M2P 2A8 Canada	Martin Buchanan Paul Clark John Maas Claude Paquette Tom Smith	(416) 229-2729 (416) 229-2729 (416) 229-2729 (416) 229-2729 (416) 229-2729
Cray Research France 7 Rue de Tilsitt 75017 Paris France	Anne Beauchamp	766 01 55
Cray Research GMBH Perhamerstrasse 31 8000 Munchen 21 West Germany	Walter Holzmaier Wolfgang Kroj	089 56014 0 089 56014 0
Cray Research, Inc. 5350 Manhattan Circle Boulder, CO 80302	Sonya Anderson Bob Biro	(303) 499-3055
Cray Research, Inc. 1100 Lowater Rd. Chippewa Falls, WI 54749	Thomas Hewitt Lou Saye Gary Shorrel	(715) 726-1211 (715) 726-1255

Cray Research, Inc.
5847 San Felipe, Suite 3000
Houston, TX 77057

Larry Stewart

(713) 975-8998

Cray Research, Inc.
1440 Northland Dr.
Mendota Heights, MN 55120

Vic Achenbach
John Aldag
Walt Anderson
Peggy Boike
Earl Bolen
Mike Booth
John Dawson
Pat Donlin
Stuart Drayton
Jean Egerman
Denise Gaertner
Brian Gaffey
Larry Gates
Chris Hector
Dick Hendrickson
Thea D. Hodge
Clay Kirkland
Dave Knaak
Bryan Koch
Lisa Krause
Loren Lemmerman
Paul Leskar
Margaret A. Loftus
Grey Lorig
Don Mason
Al Matchinsky
Jim Miller
Jim Nelson
Bob Numrich
Peter Rigsbee
Gregory Russell
Dave Sadler
Larry Schermer
Gayle F. Smith
Karen Spackman
John Stephens
Gerry Stimmler
D. Thompson
Brian Walsh
Bing Young

(612) 452-6650

Cray Research, Inc.
608 2nd Avenue South
Minneapolis, MN 55402

Mary Amiot
Bruce Kasson
Michael Mott

(612) 333-5889

(612) 333-5889

(612) 333-5889

Cray Research, Inc.
5776 Stone Ridge Mall Road
Pleasanton, CA 94566

Howard Watts

(415) 463-2800

Cray Research (UK), Ltd. Cray House London Road Bracknell, Berkshire RG12 2SY England	Martin Cutts John G. Fleming Peter Griffiths Stewart Ross	44 344 485971 44 344 485971 44 344 485971 44 344 485971
Department of National Defense Ottawa, Ontario Canada	John Mulholland	(613) 998-4183
DFVLR WT-DV D - 8031 Wessling West Germany	Peter Herchenbach	8153 28-911
E I Du Pont de Nemours E304, DuPont Co. Wilmington, DE 19898	Aaron J. Owens	(302) 772-1762
Electricite de France 1 Avenue du General de Gaulle A2-004 92140 Clamart France	Yves Souffez	3 765 4018
Environment Canada Ice Center 365 Laurier Ave. West Ottawa, Ontario K1A 0H3 Canada	Zavie Miller	(613) 996-0001
Environment Canada 2121 North Service Road Trans-Canada Highway Dorval, Quebec H9P 1J3 Canada	Bruce Attfield Raymond Benoit G.E. Berlinguette Gary Cross Jean-Francois Gagnon Mario Lepine Andre Marien Claude Payette Michel Valin	(514) 683-9414 (514) 683-9414 (514) 683-8151 (514) 683-8152 (514) 683-9414 (514) 683-7768 (514) 683-4192 (514) 683-8152 (514) 683-4525
European Centre for Medium Range Weather Forecasts Shinfield Park Reading, Berkshire RG2 9AX England	David Dent Claus Hilberg Geerd-R. Hoffmann	734-876000 734-876000 734 876000
Exxon Company 3616 Richmond Room 107 Houston, TX 77046	Brian Vohs	(713) 965-7534

Exxon Production Research Company P.O. Box 2189 Houston, TX 77252	Harry L. Brondel	(713) 940-4838
Fairchild 1801 McCarthy Blvd. Milpitas, CA 95035	Charles Dangelo	(408) 942-2587
Ford Motor Company Engineering Computer Center P.O. Box 2053 Dearborn, MI 48121	Neil St. Charles Jim Viculis	(313) 845-8493 (313) 845-8492
GA Technologies, Inc. P.O. Box 85608 San Diego, CA 92138	Sid Karin Fred McClain	(619) 455-4597 (619) 455-4597
General Motors Research 12 Mile and Mound Roads Warren, MI 48090-9055	Dean Hammond Ronald Kerry Karen M. Schaefer	(313) 575-3372 (313) 575-3208 (313) 575-3237
Government Communication Hqtrs. 1212 Priors Rd. Cheltenham, Gloucestershire GL52 5AJ England	Alan Phillips	0242521491
Grumman Data Systems Corp. 1111 Stewart Avenue Bethpage, NY 11714	Luke Kraner Don MacKenzie Paul Muzio James Poplawski John Riordan	(516) 346-2136 (516) 575-1859 (516) 575-2950 (516) 575-2934 (516) 575-7684
Institute for Defense Analyses Thanet Road Princeton, NJ 08540	Robert L. Cave Jeffrey Huskamp Helene Kulsrud Richard Schultz	(609) 924-4600 (609) 924-4600 (609) 924-4600 (609) 924-4600
KFA Juelich ZAM Nuclear Research Center Postfach 1913, D-5170 Juelich Germany	Ulrich Detert	02461 616434
Koninklijke/Shell Exploratie Produktie Laboratorium Volmerlaan 6 2288 GD Rikswijk (ZH) The Netherlands	A.E. Stormer	070 112741

Konrad-Zuse-Zentrum fur Informationstechnik Berlin Heilbronnerstrasse 10 D-1000, Berlin 31 West Germany	Hubert Busch	030 30 32 743
Lawrence Livermore National Laboratory P.O. Box 808 Livermore, CA 94550	Tim Axelrod Kent Crispin Jed Donnelley Edmund Goodwin Patrick Gray Curtis Klutts Jerry Owens Ed Schoonover Robert E. Strout II Richard Watson Mary Zosel	(415) 422-4002 (415) 422-4309 (415) 422-1259 (415) 422-4049 (415) 422-4068 (415) 422-1646 (415) 422-3767 (415) 422-4002 (415) 422-9216 (415) 422-4002
Lockheed Advanced Aeronautics Company D60-40, U50, P 2 P.O. Box 551 Burbank, CA 91520	Doug Ford Howard Weinberger	(805) 257-5720 (805) 257-5725
Lockheed Missiles and Space Corp. 1111 Lockheed Way Org 1943, Bldg. 102 Sunnyvale, CA 94086	Lee Coven Jack Sherman T.D. Telford	(408) 742-4844 (408) 742-8993 (408) 742-0948
Los Alamos National Laboratory P.O. Box 1663 Los Alamos, NM 87545	J. Wayne Anderson Christopher Barnes Richard O. Branch Ralph Brickner Ingrid Bucher Granville Chorn John Dragon Rebecca Koskela Jerry Melendez Fred J. Montoya Mark Roschke Margaret Simmons Joseph Thompson Elizabeth Williams	(505) 667-1977 (505) 667-4370 (505) 667-4890 (505) 667-8385 (505) 667-2830 (505) 667-5683 (505) 667-4812 (505) 667-8887 (505) 667-7785 (505) 667-4890 (505) 667-7073 (505) 667-1749 (505) 667-5553 (505) 667-2496
Max Plank Institut fur Plasmaphysik D8046 Garching West Germany	Ute Schneider Wolfgang Schneider	
McDonnell-Douglas P.O. Box 516 St. Louis, MO 63166	F.B. Hunt Mike Jones James McCoy	(314) 232-1938

Mobil-Mepsi Mepsi Computer Center P.O. Box 900 Dallas, TX 75221	Kevin Brewer	(214) 658-4302
NASA Ames Research Center Mail Stop 233-1 Moffet Field, CA 94035	David H. Bailey E.N. Miya James "Newt" Perdue	(415) 694-6841 (415) 694-6453 (415) 694-5189
NASA Ames Research Center 808 Burlway Rd., Suite 207 Burlingame, CA 94010	Ron Levine	(415) 342-2229
National Center for Atmospheric Research P.O. Box 3000 Boulder, CO 80307	Ann Cowley Karen Friedman Gary Jensen Walter Macintyre Richard K. Sato Sandra J. Walker	(303) 497-1223 (303) 497-1276 (303) 497-1289 (303) 497-1204 (303) 497-1287 (303) 497-1267
National Magnetic Fusion Energy Computer Center P.O. Box 5509 Livermore, CA 94550	Hans Bruijnes F. David Storch	(415) 422-4012
National Science Foundation Washington, DC 20550	John Connelly	
National Security Agency Ft. George G. Meade, MD 20755	Joseph J. Barney Claudia R. Cannady C. Thomas Myers Lynne D. Rockenbauch Gary L. Stahley W.T. Truesdale	(301) 766-1722 (301) 688-7398 (301) 730-0370 (301) 987-6042 (301) 688-6275
Naval Research Laboratory 4555 Overlook Avenue S.W. Washington, DC 20375	Harvey Brock Judith L. Flippen-Anderson Dale Pfaff Rudi F. Saenger	(202) 767-3887 (202) 767-2624 (202) 767-3190 (202) 767-2751
SAAB-Scania Aerospace Division S-58188 Linkoping Sweden	Sven Sandin	013 18 23 57
Sandia National Laboratories Albuquerque, NM 87185	Mark Kiefer Frank Mason	(505) 844-0855
Sandia National Laboratories P.O. Box 969 Livermore, CA 94550	Hilary D. Jones Gordon J. Miller Karen L. Sheaffer	(415) 422-2892 (415) 422-2964 (415) 422-3431

San Diego Supercomputer Center P.O. Box 85608 San Diego, CA 92138	Daniel Drobnis	(619) 455-4189
Schlumberger - Doll Research Old Quarry Rd. PO Box 307 Ridgefield, CT 06877	Ray Kocian	(203) 431-5522
Shell Oil P.O. Box 20709 Houston, TX 77025	James Colby C.W. Smith	(713) 795-1696 (713) 795-1696
Societe Nationale ELF Acquitaine SNEA(P) Rue Jules Ferry 64000 Pau France	Michel Morin	(59) 83 4146
SOHIO 1 Lincoln Center 5400 LBJ Freeway Dallas, TX 75240	Rex Shadrick	(214) 960-4017
Swiss Federal Institute of Technology - Lausanne Batiment du DMA Ecublens CH-1015 Lausanne Switzerland	Pierre Santschi	021 47 22 11
UKAEA Harwell Bldg. 8.12 Harwell, Oxfordshire OX11 0RA England	Michael G. Schomberg	235 24141 3263
United Information Services Co. 2525 Washington Kansas City, MO 64108	Nate Losapio	(816) 221-9700
University of Illinois 1304 W. Springfield Urbana, IL 61801	Sue Greenberg Sandy Moy	
University of London Computer Center 20 Guilford Street London WC1N 1DZ England	Christopher Lazou Dave Lexton	01 405 8400 01 405 8400

University of Toronto Computing Services 255 Huron Street Toronto, Ontario M5S 1A1 Canada	Bob Chambers Warren Jackson Edmund West	(416) 978-7092 (416) 978-8948 (416) 978-4085
US Air Force AFGWC Offut AFB, NE 68113	Rand C. Huso Joe Luteran	(402) 294-4671 (402) 294-4029
US Air Force Weapons Laboratory Kirtland AFB, NM 87117-6008	David Pelowitz Larry Rapagnani	(505) 844-9618 (505) 844-9618
Westinghouse Electric Corp. P.O. Box 355 Pittsburgh, PA 15146	Fran Pellegrino Robert Price James J. Sherin	(412) 374-4281 (412) 374-5826 (412) 374-5720
Zero One Systems, Inc. 2431 Mission College Blvd. Santa Clara, CA 95054	Kent Koeninger Paul Richards	(415) 694-6555 (408) 988-3030

CUG Site Contact List
March 1986

Adam Opel AG (OPEL CRAY)
Bahnhofplatz
Russelsheim
D-6090
Germany

Installation Delegate
T. Zimmerschied

0049-6142-663797

Air Force Weapons Laboratory (AFWL AD)
AFWL/SI
Kirtland AFB, NM 87117-6008

Installation Delegate
Larry Rapagnani

(505)844-0441

Technical and Operations Contact
Mike Gleicher

(505)844-9964

Arabian American Oil Company (ARAMCO)
EXPEC Computer Center
Dhahran, Saudi Arabia

TELEX: 601220 ARAMCO SJ

Installation Delegate
Wayne Schmaedeke
X-2660

(011)966-3-87-65155

Technical Contact
Alfred Anderson
X-2650

(011)966-3-87-61188

Operations Contact
Gene McHargue
Box 10356

(011)966-3-874-1945(or 3830)

Arnold Engineering Development Center (AEDC-CCF)
Central Computer Facility
Arnold Air Force Station, TN
37389

Installation Delegate
Larry Cunningham (615)454-7263
AEDC MS 100

Technical Contact
Wayne Neese (615)454-4294
AEDS MS 100

Atlantic-Richfield Oil & Gas Company (ARCO)
2300 Plano Parkway
Plano, TX 75075

TWX 910 861 4320

TELEX 73-2680

Facsimile Transmission DMS 1000(214) 422-3657

Installation Delegate
Dean Smith (214)422-6415
PRC - C2292

Technical Contact
B.Y. Chin (214)422-6627
PRC - 2211

Operations Contact
Chuck Murphy (214)422-6612
PRC - 5141

Atomic Energy Research Establishment (HARWELL)
Harwell, Oxfordshire
OX11 0RA, England

TELEX 83135 ATOM HA G

Installation Delegate
A. E. Taylor 0235-24141, x.3053
H 7.12

Technical Contact
Don Sadler 0235-24141, x.3227
Bldg. 8.12

Operations Contact
Michael Schomberg 0235-24141, x.3263
Bldg. 8.12

Atomic Weapons Research Establishment (AWRE)
Aldermaston
Reading, RG7 4PR
England

TELEX 848104 or 848105

Installation Delegate

L. M. Russell

07356-4111, x.6678

Technical Contact

P. A. Janes

07356-4111, x.4045

Operations Contact

M.D.P. Fasey

07356-4111, x.6491

AT&T Bell Laboratories (ATTBLMH)
600 Mountain Avenue
Murray Hill, NJ 07974

TELEX 13-8650

Facsimile (201)582-2608

(201)582-6934

Installation Delegate and Technical Contact

Peter Nelson

(201)582-6078

Operations Contact

Randolph Bell

(201)582-6368

Boeing Computer Services Company (BCS)
Post Office Box 24346
Seattle, WA 98124

Installation Delegate

Stephen Niver

(206)763-5073

MS 7A-23

Operations Contact

Jim Roetter

(206)763-5510

MS 7C-12

BP Exploration (BPLONDON)
Moor Lane
London EC2Y 9BU
United Kingdom

Installation Delegate, Technical and Operations Contact

M.P. Stanyer

(44)1-920-6156

Centre de Calcul EPFL (EPFL)
Batiment du DMA
Ecublens
CH-1015 Lausanne
Switzerland

TELEX: 25 934 EPFV CH

Installation Delegate
Pierre Santschi

021/47.22.11
011/41/21/47.22.11 (from USA)

Technical and Operations Contact
Michel Jaunin

011/41/21/47.22.02

Centre de Calcul Vectoriel Pour la Recherche (CCVR)
Ecole Polytechnique
91128 Palaiseau Cedex
France

TELEX: 691596

Installation Delegate
Tor Bloch

60 19 41 53

Technical Contact
Maurice Benoit

69 41 82 00, x.2534

Operations Contact
Paulette Dreyfus

Centre Informatique de Dorval (CID)
(Environment Canada)
2121 Trans-Canada Highway
Dorval, Quebec
Canada H9P1J3

Installation Delegate and Technical Contact
Raymond Benoit

(514)683-9414

Operations Contact
Gary Cross

(514)683-8152

Century Research Center Corporation (CRCC)
3, Nihombashi Honcho 3-chome, Chuo-ku
Tokyo, Japan 103

TELEX 252-4362 CRCNET J

Installation Delegate
Mike(Mitsuru) Maruyama (03) 665-9901

Technical Contact
Kazuyoshi Fukushima (03) 665-9901

Chevron Geosciences (CHEV-TEX)
2811 Hayes Road
Houston, TX 77082

Installation Delegate and Technical Contact
William Kimball (713)596-2520
Room 1114

Operations Contact
Juan Cruz (713)596-2523
Room 3302

Chevron Oil Field Research Company (CHEVRON)
3282 Beach Blvd.
La Habra, CA 90631

TELEX: 176967 via San Francisco

Installation Delegate and Technical Contact
Mostyn Lewis (213)694-9235

Operations Contact
John Kunselman (213)694-7029

CIFRAM (CIFRAM)
(CiSi-Framatome)
BP 24
Gif-sur-Yvette
91190
France

TELEX CISIPSC 691 597 F

Installation Delegate
Louis Bosset 69-08-42-03

Technical Contact
Philippe Van Surrell 69-08-67-05

Operations Contact
Regis Schoonheere 69-08-63-19

Commissariat a l'Energie Atomique/CEL-V (CEA-CEL)
BP 27
94190 Villeneuve St. Georges
France

Installation Delegate
Henri Dauty (1)569-96-60, x.6386

Technical Contact
Martine Gigandet (1)569-96-60, x.6184

Operations Contact
Claude Riviere (1)569-96-60, x.6484

Commissariat a L'Energie Atomique/CEV (CEAT)
Centre D'Etudes de Vaujours
Unite de Calcul
BP 7
77181 Courtry
France

Installation Delegate
Bruno Compoint (1) 868-8413

Technical and Operations Contact
Joseph Harrar (1) 868-8688

Compagnie Generale de Geophysique (CGG)
1, Rue Leon Migaux
BP 56
Massy CEDEX
91301
France

TELEX: CGGEC 692442F

Installation Delegate
Claude Guerin (6) 920.84.08

Conoco, Inc. (CONOCO)
1000 South Pine
Ponca City, OK 74603

Installation Delegate and Technical Contact
Julian Ford (405)767-3360
394 Park Building

Operations Contact
David Mohler (415)767-2813
394 Park Building

Consorzio Interuniversitario per la Gestione
Del Centro di Calcolo Elettronico dell'Italia
Nord-Orientale (CINECA)
6/3 Magnanelli
Casalecchio di Reno
40033
Bologna, Italy

Installation Delegate
Marco Lanzarini 39-51-576541

Cray Research, Inc.
608 2nd Avenue South
Minneapolis, MN 55402

Administrative Contact
Mary Amiot (612)333-5889

Technical and Operations Contact
Dave Sadler (612)452-6650

Deutsche Forschungs- und Versuchs-anstalt fur Luft-
und Raumfahrt (DFVLR)
Oberpfaffenhofen
Muncher Strasse 20
8031 Wessling
West Germany

Telephone: (0)8153/281
TELEX: 526401

Installation Delegate
Peter Herchenbach (0)8153/28954

Digital Productions (DIGIPROD)
3416 S. La Cienega Blvd.
Los Angeles, CA 90016

Installation Delegate
Gary Demos (213)938-1111

Technical Contact
Larry Yaeger (213)938-1111

Operations Contact
Gordon Garb (213)938-1111

E.I. DuPont de Nemours, Inc. (DUPONT)
Experimental Station
Wilmington, DE
19803

Installation Delegate
David Filkin

(302)772-3970

Operations Contact
James Chang
Bldg. 320

Electricite de France (EDF)
1 Avenue du General de Gaulle
A2-004
92140 Clamart
France

TELEX 270 400 F EDFERIM

Installation Delegate
Yves Souffez

(1) 765 40 18

Technical Contact
Bertrand Meyer

(1) 765 41 50 or
(1) 765 41 05

European Centre for Medium Range (ECMWF)
Weather Forecasts
Shinfield Park
Reading RG2 9AX
Berkshire, England

TELEX 847908

Installation Delegate
Geerd-R. Hoffmann

44-734-876000, x.340

Technical Contact
Claus Hilberg

44-734-876000, x.323

Operations Contact
Eric Walton

44-734-876000

Exxon Co. USA - EDPC (EXXONUSA)
3616 Richmond
Houston, TX 77046

TWX: (713) 965-7310

Installation Delegate
Michael Beddingfield (713)966-6134

Technical Contact
Brian Vohs (713)965-7534
107ST

Operations Contact
Don Smith (713)965-7514
245 ST

Exxon Production Research Company (EPRCO)
P. O. Box 2189
Houston, TX 77001

TELEX: 910-881-5579 (Answer back: USEPRTX HOU)

Installation Delegate
T.A. Black (713)965-4203
N-121

Technical Contact
J.E. Chapman (713)965-4689
N-121

Operations Contact
D.N. Turner (713)965-4407
N-180A

Fairchild (COMUN)
Gate Array Division
1801 McCarthy Blvd.
Milpitas, CA 95035

Installation Delegate
Carlos Dangelo (408)942-2587

Technical Contact
Carlos Dangelo (408)942-2680
Hassan Nosrati

Operations Contact
Hassan Nosrati

Ford Motor Company (FORD)
Engineering Computer Center
MD-1, Room 208
PO Box 2053
Dearborn, MI 48121

Installation Delegate
Neil St. Charles (313) 845-8493

General Dynamics Corporation (CF)
Data Systems Division
Central Center
PO Box 748
Fort Worth, TX 76101

TELEX: 768231

Installation Delegate and Technical Contact
M.H. Pittman (817) 777-3102
Mail Zone 1175

Operations Contact
H.D. Hollingsworth (817) 777-3238
Mail Zone 2169

General Motors Research (GM)
General Motors Technical Center
12 Mile and Mound Roads
Warren, MI 48090-9055

Installation Delegate and Operations Contact
Ronald Kerry (313) 575-3208

Technical Contact
Dean Hammond (313) 575-3372

Operations Contact
Karen Schaefer (313) 575-3237
270 R.A.N.B.

Government Communications Headquarters (GCHQ)
Priors Road
Cheltenham, Gloucestershire
GL52 5AJ
England

Installation Delegate and Technical Contact
Alan Phillips 0242-521491, x.2301
F/1210, Dept. X34C

Operations Contact
R. Medley 0242-521491, x.3185
F/1208

Grumman Data Systems (GDS)
1111 Stewart Avenue
Bethpage, NY 11714

Installation Delegate and Technical Contact

James Poplawski (516)575-2934
MS B34-111

Operations Contact

Steven Hornacek, Jr. (516)575-4273

Institute for Defense Analyses (IDA)
Thanet Road
Princeton, NJ 08540

Installation Delegate and Operations Contact

Robert Cave (609)924-4600

Technical Contact

Helene Kulsrud (609)924-4600

KFA Julich (KFA)
Postfach 1913
5170 Julich 1
West Germany

TELEX: 833556 KFA D

Installation Delegate

Friedel Hossfeld 02461-61-6402

Technical and Operations Contact

L. Wollschlaeger 02461-61-6420

Koninklijke/Shell Exploratie & Productie Laboratorium (KSEPL)
Volmerlaan 6
2288 GD Rijswijk (Z.H.)
The Netherlands

TELEX KSEPL NL 31527

Installation Delegate and Technical Contact

A.E. Stormer 070-112741
LS-219

Operations Contact

A.A.H. Kardol 070-112601
LS-208

Konrad Zuse-Zentrum fur Informationstechnik Berlin (BERLIN)
Heilbronnerstrasse 10
D 1000 Berlin 31
West Germany

TELEX: 183798

Installation Delegate
Jurgen Gottschewski (030)-3032-233

Lawrence Livermore National Laboratory (LLNL)
PO Box 808
Livermore, CA 94550

TWX 910 386 8339 UCLLL LVMR

Installation Delegate
Richard Zwakenberg (415)422-3750
L-300

Technical Contact
Patrick H. Gray (415)422-4049
L-60

Operations Contact
Pierre Du Bois (415)422-4007
L-67

Lockheed Advanced Aeronautics Company (XMP24110)
Dept. 60-40, Unit 50, Plant 2
PO Box 551
Burbank, CA 91520

Installation Delegate and Technical Contact
Howard Weinberger (805)257-5725

Operations Contact
Doug Ford (805)257-5720

Lockheed Missile and Space Co. (LOCKHEED)
1111 Lockheed Way
Sunnyvale, CA 94086

TELEX: 346409

Installation Delegate
Jack Sherman (408)742-8993

Technical Contact
Doug Telford (408)742-0948

Operations Contact
Jerry Roninger (408)742-5831

Los Alamos National Laboratory (LANL)
P. O. Box 1663
Los Alamos, NM 87545

Installation Delegate

Charles Slocomb (505)667-5243
MS B294

Technical Contact

Margaret Simmons (505)667-1749
MS B265

Christopher Barnes (505)667-5000
Group X-1, MS E531

Operations Contact

Tom Trezona (505)667-4890
MS 252

Max Planck Institute fur Plasmaphysik (MPI)
8046 Garching
Bei Munchen
West Germany

TELEX 05/215 808

Installation Delegate and Technical Contact

Johann Gassmann 089-3299-340

McDonnell-Douglas Corporation (MDC)
PO Box 516
St. Louis, MO 63166

Facsimile Transmission: (314)233-6149

Installation Delegate

James R. McCoy (314)233-3425
Dept. W512 - 306/3

Technical Contact

James Miget (314)234-3326
W532 - 306/3/395

Operations Contact

F. Brian Hunt (314)233-4900
W270 - 306/2E/290

Merlin Profilers Limited
1 Duke Street
Woking, Surrey
United Kingdom

Installation Delegate
Paul Blundell

Technical Contact
Andy Wright

Mitsubishi Research Institute, Inc. (MIRI)
2-3-6, Otemachi
Chiyoda-ku
Tokyo, Japan 100

TELEX 222-2287 MRI J

Installation Delegate
Nobuhide Hayakawa (03) 270-9211

Technical and Operations Contact
Shuichi Yamagishi (03) 270-9211

Mobil Exploration & Producing Services, Inc. (MEPSI)
PO Box 900
Dallas, TX 75221

Installation Delegate
Beverly Jackson (214)658-4409

MOD (P.E.), RARDE (RARDE)
Fort Halstead
Sevenoaks, Kent, TN14 7BP
England

TELEX: 95267

Installation Delegate and Technical Contact
Bob Youldon 0732-55211, x.3086
Bldg. 511

NASA/Ames Research Center (NAS)
NAS Projects Office
Moffett Field, CA 94035

Installation Delegate
John Barton (415)694-6837
MS 233-1

NASA/Lewis Research Center (NASA/LE)
21000 Brookpark Road
Cleveland, OH 44135

Installation Delegate

William McNally
MS 142-2

(216)433-4000, x.6650

National Cancer Institute (FCRF)
Frederick Cancer Research Facility
Advanced Scientific Computing Laboratory
PO Box B
Frederick, MD 21701

Installation Delegate

Charles Crum

(301)695-2765

Technical Contact

Jacob Maizel

(301)695-2532

Operations Contact

Steve Karwoski

(301)695-2775

National Center for Atmospheric Research (NCAR)
P. O. Box 3000
Boulder, CO 80307

TELEX 45694

Installation Delegate

Bernie O'Lear

(303)497-1268

Technical Contact

Eugene Schumacher

(303)497-1264

Operations Contact

Gary Jensen

(303)497-1289

National Magnetic Fusion Energy
Computer Center (NMFEC)
P. O. Box 5509, L-561
Livermore, CA 94550

TELEX 910-386-8339

Installation Delegate

Hans Bruijnes

(415)422-4012

Technical Contact

F. David Storch

(415)422-4004

Operations Contact

Marilyn Richards

(415)422-4397

National Security Agency (NSA)
Ft. George G. Meade, MD 20755

Installation Delegate
Bruce Steger (301)688-6275
T335

Technical Contact
C. Thomas Myers (301)688-6275
T335

Operations Contact
Richard W. Ader (301)688-6198
T152

Naval Research Laboratory (NRL)
4555 Overlook Avenue S.W.
Washington, DC 20375

Installation Delegate
Harvey Brock (202)767-3886

Nissan Motor Company (NISSAN)
Nissan Technical Center
560-2, Okatsukoku
Atsugi, Kanagawa
243-01
Japan

Telex: J47980

Installation Delegate, Technical and Operations Contact
Mizuho Fukuda 0462-47-5523
Engineering Computer Applications Section No. 1
Product Development Systems Department

NTT Electrical Communications Laboratories (NTT)
Nippon Telegraph and Telephone Corporation
3-9-11 Midori-cho
Musashino city, Tokyo 180
Japan

Installation Delegate
Toshimasa Suzuki (011) 81-0422-59-3001

Technical Contact
Mikio Sasaki (011) 81-0422-59-2261
Information Processing Services Section
Engineering Department

Operations Contact
Hideaki Maeda (011) 81-0422-59-3845
Information Processing Services Section
Engineering Department

ONERA - Calculateur Aeronautique (ONERA)
BP 72
Chatillon Sous Bagneux
92322
France

TELEX: ONERA 260 907F

Installation Delegate
Jean-Pierre Peltier (1) 6571160, x.2094

Technical Contact
Daniel Colin (1) 6571160, x.3098

Operations Contact
Jean Erceau (1) 6571160, x.2465

Phillips Petroleum Company
418 Information Systems Bldg.
Bartlesville, OK 74004

Installation, Technical and Operations Contact
Arvin Todd (918)661-6426

Rechenzentrum der Universitat Stuttgart (RUS)
Pfaffenwaldring 57
7000 Stuttgart 80
West Germany

TELEX: 07255445

Installation Delegate
Walter Wehinger 0711-685-5391

Rockwell International Information Systems Center (RI)
PO Box 2515
Mail Code SH10
Seal Beach, CA 90740

TELEX: 910-341-6801 (ROCK ISCW SLBH)

Installation Delegate and Technical Contact
Abraham Levine (213)594-2740

Operations Contact
Joe Henderson (213)594-2283

Royal Aircraft Establishment (RAE)
Bldg. R16
Farnborough, Hants
GU14 6TD
England

TELEX: 858134

Installation Delegate
J.M. Taylor (0252)24461, x.3042

Technical Contact
D. Swan (0252)24461, x.2714

Operations Contact
L. Shepherd (0252)24461, x.2375

SAAB-Scania (SAAB)
Aircraft Division
S-58188 Linkoping
Sweden

TELEX: 50040 SAABLGS

Installation Delegate and Technical Contact
Sven Sandin 4613 182357

Operations Contact
Stig Logdberg 4613 182371

Sandia National Laboratories (SNLA)
Albuquerque, NM 87185

Installation Delegate
Melvin Scott (505)844-4075
Department 2641

Technical Contact
Frank Mason (505)844-2332
Division 2641

Operations Contact
Kelly Montoya (505)844-1234
Department 2630

Sandia National Laboratories (SNLL)
PO Box 969, East Avenue
Livermore, CA 94550

Installation Delegate and Technical Contact
Dona Crawford (415)422-2192
D8235

Operations Contact
M.H. Pendley (415)422-2965
D8236

San Diego Supercomputer Center (SDSC)
PO Box 85608
San Diego, CA 92138

TELEX: 695065

Installation Delegate and Technical Contact
Fred McClain (619)455-4597

Operations Contact
Dan Drobnis (619)455-4189

Schlumberger-Doll Research (SCHLUMBE)
Old Quarry Road
PO Box 307
Ridgefield, CT 06877

TELEX: 643359

Installation Delegate
Bob Snow (203)431-5527

Technical Contact
Raymond Kocian (203)431-5522

Operations Contact
Josephine Murray (203)431-5524

Shell Oil Company (SHELLOIL)
PO Box 20709
Houston, TX 77025

TELEX: 71-378-7530 (answer back - Shell MTM HOU)

Installation Delegate
L.J. Kealy (713)795-3320

Technical Contact
B.D. Huff (713)795-3193
Rm. 5B48

Operations Contact
C.W. Smith (713)795-1696
Rm. 1P10

Shell U. K. (SHELLUK)
Rowlandsway Wythenshawe
Manchester M22 5SB
United Kingdom

TELEX: 668613

Installation Delegate
David Cheater 061-499-4357

SNEA (ELF)
Rue Jules Ferry
Pau 64000
France

TELEX: Petra 560 804F

Installation Delegate, Technical and Operations Contact
Michel Morin 59-834146

SOHIO
Geophysical Data Center
1 Lincoln Center
5400 LBJ Freeway
Suite 1200-LB 25
Dallas, TX 75240

Installation Delegate, Technical and Operations Contact
Mark Rehrauer (214)960-4336

SVERDRUP Technology, Inc. (SVERDRUP)
Arnold Air Force Station, TN 37389

Installation Delegate and Operations Contact
John L. Roberson (615)455-2611, x-5294
ASTF MS900

Toshiba Corporation (TIS)
TIS Division
1-1, Shibaura
Minato-Ku, Tokyo, 105
Japan

TELEX: J22587

Installation Delegate
Kenjo Yoshimura 044-541-1743

Technical and Operations Contact
Kyosuke Tsuruta 044-541-1743

United Information Services, Inc. (UISCO)
2525 Washington
Kansas City, MO 64108

Installation Delegate and Operations Contact
Nate Losapio (816)221-9700, x.6535

Technical Contact
John McComb (816)221-9700

University of Illinois at Urbana-Champaign (UIUCNCSA)
National Center for Supercomputing Applications
1011 W. Springfield
Urbana, IL 61801

Installation Delegate and Technical Contact
Win Bernhard (217)333-8049

Operations Contact
Mike Smith (217)244-0708

University of London (ULCC)
Computer Center
20 Guilford Street
London WC1N 1DZ
England

TELEX: 8953011

Installation Delegate
Richard Field (01)4058400

Technical Contact
Harald Kirkman

Operations Contact
Lawrie Tweed

University of Minnesota Computer Center (MINN)
2520 Broadway Drive
Lauderdale, MN 55113

Installation Delegate
John Sell (612)373-7878

Technical Contact
Linda Gray (612)376-5603

Operations Contact
Elizabeth Stadther (612)373-4920

University of Texas System (UTXCHPC)
Center for High Performance Computing
Commons Building, Balcones Research Center
PO Drawer S
Austin, TX 78713-7388

Installation Delegate
Charles Warlick (512)471-2472

Technical Contact
William Bard (512)471-2472

Operations Contact
Robert Baker (512)471-2472

University of Toronto (UTORONTO)
Computing Services
255 Huron St.
Toronto, Ontario M5S1A1
Canada

Installation Delegate and Technical Contact
Edmund West (416)978-4085
MPP 331

Operations Contact
Robert Chambers (416)978-7092
MP 350

Westinghouse Electric Corporation (WESTESCC)
Energy Systems Computer Center
P. O. Box 355
Pittsburgh, PA 15146

TELEX: 234992503 USA

Installation Delegate
Robert Price (412)374-5826
MNC 206E

Technical Contact
Jerry Kennedy (412)374-4399
Nuclear Center 180

Operations Contact
R.W. Kunko (412)374-4674
Fran Pellegrino

ZeroOne Systems, Inc. (ZERO)
2431 Mission College Blvd.
Santa Clara, CA 95054-1297

Installation Delegate, Technical and Operations Contact
Glenn Lewis (408)988-3030

CUG
CALL for PAPERS
Garmisch-Partenkirchen, West Germany
Fall 1986

NAME: _____

ORGANIZATION: _____

ADDRESS: _____

TELEPHONE: () _____

TITLE OF PAPER: _____

TWO OR THREE SENTENCE ABSTRACT: _____

EQUIPMENT REQUIRED OTHER THAN 35MM SLIDE PROJECTOR OR
OVERHEAD PROJECTOR:

SUGGESTED SESSION:

GENERAL SESSION	___	LANGUAGES	___
I/O	___	LIBRARIES	___
FRONT ENDS	___	MULTITASKING	___
NETWORKING	___	OPTIMIZATION	___
OPERATIONS	___	PERFORMANCE EVALUATION	___
OPERATING SYSTEMS	___	APPLICATIONS	___
GRAPHICS	___	DATA MANAGEMENT	___

OTHER _____

RETURN BY JULY 1, 1986 TO:

David Lexton
University of London
Computer Center
20 Guilford Street
London WC1N 1DZ England

I Want to

HELP CUG

By working on the

CRAY Operating Systems SIG _____
CTSS SIG _____
Networking and Front Ends SIG _____
Languages and Libraries SIG _____
Operations SIG _____
Performance Improvement SIG _____
Hold a CUG Meeting at my Site _____
Lead a Workshop on _____
Other _____

NAME _____

ADDRESS _____

TELEPHONE _____

Please complete this form if you wish to assist with CUG meetings.

Return to:

Karen Friedman
NCAR
P.O. Box 3000
Boulder, Colorado 80307
USA



Vertical line on the left side of the page.

