CSRD Rpt. No. 684

~~DOE/ER/25001--T1~~
DOE/ER/25001--T1
Received
DEC 09 1987

DOE/ER/25001--T1
DE88 003564

**CENTER FOR SUPERCOMPUTING
RESEARCH AND DEVELOPMENT**

QUARTERLY REPORT

FIRST QUARTER 1987

STAFF

June 1987

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Center for Supercomputing Research and Development
University of Illinois
305 Talbot - 104 South Wright Street
Urbana, IL 61801-2932
Phone: (217) 333-6223

This work was supported in part by the National Science Foundation under Grant Nos. US NSF DCR85-09970, US NSF DCR84-06916, and US NSF DCR84-10110, and the U. S. Department of Energy under Cont. No. US DOE-DE-FG02-85ER25001, the Air Force Office of Scientific Research under Grant No. AFOSR-85-0211, the IBM Donation, Alliant Computer Systems, Apollo, Control Data, and Digital Equipment.

MASTER
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

TABLE OF CONTENTS

HARDWARE QUARTERLY PROGRESS REPORT	1
1. HARDWARE DEVELOPMENT	1
1.1 Global Interface Board (C.-Q. Zhu)	1
1.2 Global Network Board (J. Konicek)	1
1.3 Global Memory Board (A. Veidenbaum)	1
1.4 Calay Designs (M. Haney)	1
1.5 Cedar Physical Packaging (N. Warter)	1
1.6 Performance Monitor Subsystem (D. Lavery)	2
1.7 Cedar Diagnostic Hardware (K. Konigsfeld, M. Haney)	2
1.8 Cedar Diagnostic Software (M. Haney, J. Szczypek)	2
1.9 CADAM Software (M. Haney)	2
2. ARCHITECTURE RESEARCH	2
2.1 Sparse Linear Systems Algorithms (E. Davidson, T. Davis)	2
2.2 A New Parafrase Pass (H. Cheong, A. Veidenbaum)	2
2.3 Data Synchronization (P. Tang, P.-C. Yew, C.-Q. Zhu)	3
2.4 Data Level Synchronization for Loops Containing Procedure Calls (Z. Li, P.-C. Yew)	3
2.5 Caches and Data Prefetching in Multiprocessors (D. Lawrie, R. Lee, P.-C. Yew)	3
2.6 Multiprocessor Cache Studies (H. Cheong, A. Veidenbaum)	3
2.7 Exploiting Locality in Lisp Programs (E. Davidson, H.-S. Sung)	3
2.8 Parallel Garbage Collection on a Virtual Memory System (S. Abraham and J. Patel: Com- puter Systems Group, CSL)	4
2.9 Comparison of Vector and Superscalar Processor Performance (E. Davidson, J.-H. Tang)	4
SOFTWARE QUARTERLY PROGRESS REPORT	

MASTER

1. OPERATING SYSTEM DEVELOPMENT	6
1.1 Xylem Virtual Memory System Development (P. Emrath, R. McGrath)	6
1.2 Xylem Booter Development (R. Barton, P. Emrath)	6
1.3 Exception Handling (P. Emrath)	7
2. OPERATING SYSTEM RESEARCH	7
2.1 Multiprocessor Run-Time Scheduling (A. Ray Miller)	7
2.2 Mass Storage Systems (G. Heuring)	7
2.3 Index (P. Emrath, D. Wright)	7
2.4 Synchronization (D. Jayasimha and M. Loui: Dept. of Electrical and Computer Engineering)	8
2.5 I/O Performance Enhancement (P. Emrath, T. McDaniel, D. Padua)	8
3. CEDAR FORTRAN	8
3.1 Cedar Fortran (J. Hoeflinger)	8
3.2 Cedar Fortran Programmer's Manual (M. Guzzi)	9
3.3 Runtime Libraries (M. Guzzi)	9
3.4 Processor Self-scheduling and Microtasking (M. Guzzi)	9
3.5 Kap (M. Guzzi)	9
3.6 Compiler Development (S. Midkiff)	9
4. SYMBOLIC PROCESSING	9
4.1 Lisp (W. Harrison, D. Padua)	9
4.2 Parallel Prolog Interpreter (D. Sehr and L. Kale: Dept. of Computer Science)	10
5. COMPILER RESEARCH	10
5.1 Data Location (S. Karlovsky, D. Padua)	10
5.2 Debugging (T. Allen, D. Padua)	10

5.3 Symbolic Algebra (D. Padua, P. Petersen)	10
5.4 Functional Programming Language (S. Ghosh)	10
5.5 Parafrase II (M. Girkar, B. Leung, C. Polychronopoulos, D. Schouten)	11
6. SCIENTIFIC WORKSTATION ENVIRONMENT: FAUST	11
6.1 Faust Editor (Y. Gaur, V. Guarna, D. Jablonowski)	11
6.2 Faust Debugger (J. Krause, P. Lewis, D. Reed)	11
6.3 Faust Expert System (M. Berry, S. Bhansali, M. Harandi)	11
6.4 Parallel Ray Tracing (D. Defend, D. Gannon, A. Tuchman)	12
6.5 Automatic Error Analysis and Faust (B. Bliss, E. Gallopoulos)	12
APPLICATIONS QUARTERLY PROGRESS REPORT	13
1. NUMERICAL LIBRARY	13
1.1 Solving Dense and Banded Linear Systems	13
1.1.1 Iterative Methods for General Sparse Linear Systems (E. Anderson, Y. Saad)	13
1.1.2 BLAS3 Primitives (K. Gallivan, W. Jalby, U. Meier)	13
1.1.3 High-level Modules (K. Gallivan, W. Jalby, U. Meier)	13
1.1.4 Direct Schemes for Solving Block-Tridiagonal Systems (M. Berry, A. Sameh)	13
1.1.5 An Iterative Scheme for Solving Block Tridiagonal Systems (R. Bramley)	14
1.1.6 Narrow Banded Solvers (C. Ulrich, W. Harrod)	14
1.1.7 Block Reduction to Tridiagonal and Hessenberg Forms (W. Harrod)	15
1.2 Eigenvalue Problem	15
1.2.1 Singular Value Decomposition (SVD) (M. Berry, A. Sameh)	15
1.2.2 Tridiagonal Eigenvalue Problem (C. Baucom, D. Sorensen)	
1.2.3 Trace Minimization Algorithm for the Generalized Eigenvalue Problem (W. Harrod, A. Sameh)	15

1.3 Conjugate Gradient Algorithms (W. Jalby, U. Meier, A. Sameh)	15
1.4 Nonlinear Optimization and the Solution of Nonlinear Equations	15
1.4.1 Nonlinear Equation Solver (R. Guo, Y. Saad)	15
1.4.2 Numerical Conformal Mapping Method (M.-Q. Chen, A. Sameh)	16
1.4.3 Partially Separable Large-scale Optimization (M.-Q. Chen, S.-P. Han)	16
1.5 Rapid Elliptic Solvers (E. Gallopoulos, Y. Saad)	16
1.6 Kalman Filtering (K. Gallivan)	16
1.7 Strategies for Cache and Local Memory Management by Global Program Transformation (K. Gallivan, D. Gannon, W. Jalby)	17
2. APPLICATIONS	17
2.1 Circuit Simulation (K. Gallivan, Y.-C. Ju, P. Koss, S. Lo, R. Saleh, D. Sorensen, X. Wang, E. Xia, G.-C. Yang)	17
2.1.1 Nonlinear and Waveform Relaxation	17
2.1.2 Direct Methods	18
2.2 Computational Chemistry (A. Galick, W. Harrod)	18
2.3 Computational Fluid Dynamics	18
2.3.1 Computational Fluid Dynamics (R. Roloff, B. Semeraro)	18
2.3.2 Convection-Diffusion P.D.E. Solver (J. Scroggs, D. Sorensen and R. Chin and G. Hedstrom: Lawrence Livermore National Laboratory)	19
2.3.3 Navier-Stokes Solver (W. Jalby, U. Meier and L. Hageman: Westinghouse)	19
2.4 Structural Mechanics and Dynamics	19
2.4.1 Plate Bending (H.-C. Chen)	19
2.5 Weather Simulation (E. Gallopoulos, K.-S. Tai)	19
2.6 Applications Graphics (B. Semeraro, A. Tuchman)	20
PERFORMANCE EVALUATION PROGRESS REPORT	21

1. Performance Evaluation Tools (A. Malony)	21
2. Performance Evaluation (K. Gallivan, A. Malony, W. Jalby, Y. Saad)	21

HARDWARE QUARTERLY PROGRESS REPORT FIRST QUARTER 1987

1. HARDWARE DEVELOPMENT

1.1 Global Interface Board (C.-Q. Zhu)

GIB has been modified from version GIB.1 to GIB.1A. There are 8 GIB.1A's in each cluster of Cedar.1A. Each GIB.1A is connected to one of the eight CE's in a cluster of the Cedar.1A (there are 4 CE's in each cluster of Cedar.1). The redesign combined with the improved placement of the PC board reduces the final routing length of the board. The GIB.1 is presently in fabrication. A preliminary patent application has been written for the prefetch implementation of the GIB.

1.2 Global Network Board (J. Konicek)

GNBI has been redesigned to reduce the routing problem of this board. ASI has placed the board to completion and is now in the process of routing the board. Preliminary routing reports look promising.

A new board type, GNBK, has been designed for use in Cedar 1 and 1A. GNBK converts the single-end backplane signals produced by GNBI to differential cable signals. No switch function is provided by GNBK. GNBK does provide one level of buffering (either latches or registers). GNBK will replace the very-risky GNBO. Because of the buffer selection on GNBK, it should not reduce performance of the network.

GNBK has been designed and the netlist with schematics has been sent to ASI. Placement of this board should begin shortly. GNBK is expected to be a relatively easy board development because it uses thru-hole technology and has only 350 EIC on the same form factor as GNBI.

1.3 Global Memory Board (A. Veidenbaum)

The layout and routing of the Global Memory board has continued as an interactive process with ASI, Inc. A preliminary route has been completed and analyzed. Based upon this, several corrections have been made and the routing strategy adjusted. ASI is presently cleaning up the final route.

1.4 Calay Designs (M. Haney)

The layout and routing for the Global Memory Emulator board has been completed on the Calay printed circuit board design workstation, and the board has been sent out for fabrication. This board will be used in initial debugging of the Cedar system. This board marks the 9th Cedar-related Calay project completed in as many months since the Calay system was assigned to handle the many secondary-support hardware designs required by the Cedar system.

Two different Global Network Backplane designs have been completed and sent to printed circuit fabrication houses for price quotation. As a consequence of the cost estimates, a third Network Backplane design was created, and has been released for fabrication. Work is near completion on the Global Memory Backplane design.

The database for the Alliant backplane was obtained from CAD Services, and it was established that the backplane could be stretched to accommodate the four additional GIB slots for Cedar1.A. Work is underway to establish a consistent database that includes the original Alliant design as well as the Cedar-specific modifications.

1.5 Cedar Physical Packaging (N. Warter)

An Alliant FX/8 has been dismantled and installed in the left Cedar bay. This Alliant is fully operational and is now supporting users. This bay is being monitored by a temporary environment control board. Work is continuing on the final PC control board which will monitor the entire Cedar environment: airflow, temperature, humidity, and power. The monitoring board will sample sensor data, relay sensor information and out of range warnings over an ethernet connection to an IBM PC/AT. It will

remove power from the cabinets if sensors exceed the maximum safety limit.

1.6 Performance Monitor Subsystem (D. Lavery)

PCB layout was completed on the Scaler with Data Storage Module. Four copies of the board were manufactured at Pentaplex and one was loaded by the Physics department. This board has been debugged and is fully operational. A Trace Storage Module has been designed. Each module is capable of storing traces of an 8 bit word.

1.7 Cedar Diagnostic Hardware (K. Konigsfeld, M. Haney)

The development of the diagnostic network's operating system, LANOS, continued during the first quarter of 1987. With one exception, all of the code modules have been written. Several of them are now tested and operational. Software is now being developed is on the CSRD-designed IBM PC/AT controller board using the in-circuit-emulator for runtime debugging. Currently, the entire diagnostic development system is being documented.

1.8 Cedar Diagnostic Software (M. Haney, J. Szczyppek)

Fault detection and location in the Cedar interconnection network is being investigated. The goal is to develop diagnostic software which can do the fault detection/location without necessitating hardware reconfiguration of the network. Preliminary work is being done with respect to the diagnostic processor software and its interface with the GIB and GMB. Work is continuing on development of the system exerciser software which fully saturates the network connection paths.

1.9 CADAM Software (M. Haney)

The mechanical engineering portion of the CADAM computer aided design software has been installed on the IBM mainframe at CSRD. Preliminary experimentation has indicated that the software is fully operational. The electrical engineering portion of the software is currently being installed. The CADAM software package is a university grant from CADAM, Inc., and is anticipated to play a significant role in the Cedar 2 design effort.

2. ARCHITECTURE RESEARCH

2.1 Sparse Linear Systems Algorithms (E. Davidson, T. Davis)

PBit is a new version of PSolve that replaces the column index vectors in PWork with a bit matrix. The bit matrix representation uses a single bit for each element of the A matrix; a bit is on if the element is nonzero and off otherwise. This allows use of the Alliant compress/expand instructions. The nonzero element values are stored in the same way as PWork, and the underlying algorithm is unchanged. PBit is about twice as fast as in PWork. The motivation behind the bit matrix representation is to allow PSolve to run in vector mode on the Alliant and to improve pivot selection to reduce fill-in.

One problem with current PSolve variants is the limitation on the system size. Currently, the nonzero values of each row are compacted into a single memory array that has enough space left over to allow for maximal fill-in. In other words, the compact representation saves computation, but wastes memory space. A dynamic memory allocator is being developed.

A preliminary patent application has been written for the hardware align unit used by the PVector version of PSolve.

2.2 A New Parafrase Pass (H. Cheong, A. Veidenbaum)

A pass called RN1CRU has been added to Parafrase. RN1CRU converts an rn1 loop with a loop bound of N into a doall loop (with a bound of N/P) and an rn1 loop (with a loop bound of P). The addition of this pass makes realistic trace generation possible for R(N,1)s.

2.3 Data Synchronization (P. Tang, P.-C. Yew, C.-Q. Zhu)

Work has been started on data synchronization for parallel loops. The existing data dependence tests are being examined and compared in terms of completeness and complexity. An effort to improve the exhaust test proposed by Zhu and Yew [ZhYe84] is underway. The problem of data synchronization will be studied from the perspective of processor self-scheduling.

2.4 Data Level Synchronization for Loops Containing Procedure Calls (Z. Li, P.-C. Yew)

When there are cross-iteration data dependences, a loop can still be executed concurrently on shared memory multiprocessors. In many cases, to enforce these data dependences, data level synchronization is very useful. In order to perform data level synchronization, the set of iterations which reference the same data element and cause data dependences need to be identified exactly. A test scheme is being formulated to obtain exact synchronization patterns at compile time. This preliminary study is necessary for research into synchronization problems for loops containing procedure calls.

2.5 Caches and Data Prefetching in Multiprocessors (D. Lawrie, R. Lee, P.-C. Yew)

The cache block size in multiprocessors was considered by examining two additional aspects of cache performance: i) the scheduling of parallel loop iterations on limited processors and ii) the method in which cache blocks are loaded into the cache. Blocking the parallel iterations in a loop to improve the spatial locality of reference and using read-through to minimize cache block latency tend to favor a larger block size. However, even with these factors, when all the characteristics are modeled, a block size of 8 words or less is optimal.

Two types of software cache coherence schemes have been compared. *Data marking*, where data is marked for the duration of the program as either cacheable or non-cacheable was compared to *reference marking*, where references are marked to allow data to change cacheability within one program. Several reference marking schemes have been developed and have been shown to outperform data marking schemes.

2.6 Multiprocessor Cache Studies (H. Cheong, A. Veidenbaum)

Additional simulations were performed to verify the results reported earlier. The conclusions about the effect of block size, number of processors, and hit ratio for multiprocessors were confirmed.

The current study concerns:

- (1) the effect of cache size on speedup and hit ratio;
- (2) performance improvement due to the data cache alone;
- (3) different ways of measuring hit ratios in multiprocessor systems and how they can be used to describe the system performance; and
- (4) the effect of cache invalidation and parallel execution of programs on cache hit ratios.

2.7 Exploiting Locality in Lisp Programs (E. Davidson, H.-S. Sung)

It is expected from the Clark¹ data that prefetching line $i-1$ as well as line $i+1$ will reduce the cache miss ratio of Lisp execution even further, because Lisp pointers point not only to forward but also to backward neighbors. Tracing facilities and a cache performance analyzer have been developed to study this

¹Bobrow, D.G. and D.W. Clark, *Compact Encodings of List Structure*, ACM Trans. on Programming Languages and Systems, Vol. 1, No. 2, October, 1979.

²Clark, D.W. and C.C. Green, *An Empirical Study of List Structure in Lisp*, Communications of ACM, Vol. 20, No. 2, February, 1977.

problem further. Preliminary tests of some Lisp programs, in Franz Lisp on UNIX, showed that this prefetching scheme did help to achieve a higher cache hit ratio with less total traffic increase than achieved by Smith's³ scheme.

To strengthen the linear property and to take advantage of pointer locality, we propose using a co-processor between the CPU (Lisp processor) and memory to execute both list processing and garbage collection in parallel. The co-processor, together with a pair of FIFO buffers, can perform both prefetching and copying garbage collection. The CPU can thus be dedicated to Lisp execution without being periodically interrupted to perform garbage collection work. With this kind of parallel prefetching, linearization, and garbage collection, memory wait time is expected to be reduced further, thereby achieving higher system performance. A Lisp interpreter, in C on UNIX, is being developed to evaluate this design.

2.8 Parallel Garbage Collection on a Virtual Memory System (S. Abraham and J. Patel: Computer Systems Group, CSL)

In [AbPa87], an architecture and an associated algorithm for parallel garbage collection on a virtual memory system are described. Garbage collection involves the identification of memory cells that have been released by application programs involving list processing. Artificial intelligence (AI) architectures must support efficient garbage collection because AI applications are often programmed in list processing languages.

The proposed scheme differs from existing schemes in two respects. Most parallel garbage collection algorithms are designed for shared memory machines which have certain implicit synchronization functions on variable access. The proposed algorithm is designed for virtual memory systems where both the list processor and the garbage collector have private memories. The enforcement of coherence between the two private memories can be expensive and is not necessary in this scheme. Secondly, unlike other algorithms, the proposed algorithm does not attempt to collect cells released by the list processor during the current garbage collection cycle. Consequently, the list processor and the collector need to synchronize only at the beginning and end of a cycle, and the list processor incurs little overhead.

2.9 Comparison of Vector and Superscalar Processor Performance (E. Davidson, J.-H. Tang)

The architectural drawbacks of vector machines have been identified and several scalar architectures have been studied to determine their capability in executing both vectorizable and nonvectorizable codes up to the full concurrency allowed by program kernels.

Vector machine drawbacks can be seen in: functional unit utilization, processor to memory traffic, and processor register efficiency. A model for performance comparisons between vector and scalar machines is being developed. Several problematic examples are mentioned below.

A scalar machine with specialized logic to achieve effective data forwarding from memory is able to execute nonvectorizable loops concurrently at speeds limited only by data dependence. For vectorizable loops, a scalar processor with one addition pipeline unit and one multiplication pipeline unit, such as the Cray X-MP has, can fully utilize at least one of the two units after some initial transient. But even with flexible chaining, the Cray X-MP can not fully utilize either of the two functional units for the following loop:

```
DO 20 i = 1, N
  P(i) = U(i) + V(i)
  Q(i) = P(i) * W(i)
  Y(i) = s * P(i) + (P(i) + Q(i)) * Q(i)
20 continue
```

³Smith, A.J., *Cache Evaluation and the Impact of Workload Choice*, The 12th Annual International Symposium on Computer Architecture, June 1985.

When array indices misalign in a vectorizable loop, e.g. both $V(i)$ and $V(i+1)$ are accessed in the same loop iteration, the traffic of vector processor to memory and the loop execution time are both increased since the V vector is read from memory twice and uses two vector registers. For most vectorizable loops when memory bandwidth is the major bottleneck such as in the Cray-1 vector processor, the loop execution time is limited by the memory access time.

Register spilling causes severe performance degradation in register-register vector processors. However, in-processor data storage of scalar processors can be reconfigured for different modes of operation, not only to eliminate register spilling, but also to enable the compiler to apply loop coalescing and loop fusion intensively to reduce loop overhead.

SOFTWARE QUARTERLY PROGRESS REPORT FIRST QUARTER 1987

1. OPERATING SYSTEM DEVELOPMENT

1.1 Xylem Virtual Memory System Development (P. Emrath, R. McGrath)

The concept of "Shared Cluster" was considered in detail and new specifications developed. Shared Cluster memory is shared by all tasks of a Xylem process, but is "cached" in cluster memory. If this memory is modified, then maintaining coherence of all the local copies can be a problem. Previous versions of Xylem did not allow "Shared Cluster" memory to be modified, but several applications may be able to benefit from modifiable and "cached" memory. The problems presented by this were considered and a new system call was added to allow user programs or libraries to maintain the coherence of such memory.

The *memctl()* system call allows the attributes, location, and availability of memory to be manipulated as the program runs. A page may be accessible for reading, writing, and/or execution. A page of memory may be located in Cedar global memory or local cluster memory. These attributes may be changed for process-shared or task-private memory through an appropriate system call. In addition, a special system call is provided to "invalidate" a local copy of a shared page. These calls are used to keep Shared Cluster memory coherent.

This work was seen to make the previous work on Xylem heaps obsolete, as the concept of one "Shared Global" and one "Shared Cluster" heap makes no sense when the location of memory can be changed on a page by page basis. The *memctl()* call was extended to include requests to allocate and deallocate memory with various attributes. These calls will completely replace the heap-management system calls implemented last quarter.

1.2 Xylem Booter Development (R. Barton, P. Emrath)

The booter has been enhanced such that it should now be able to boot a multi-cluster Cedar. This booter really consists of two executable files, both of which will typically reside on a floppy. The first file is the same as the Concentrix booter and will boot Diagnostix, Concentrix, or Xylem depending on which boot flags are set. When booting Xylem, the second executable file is what is really loaded by the booter. This second file is referred to as the Xylem startup code.

The Xylem startup code runs on IP0 of the cluster that is booting and starts server code running on each of the CEs of the cluster. The IP then sends commands to each of the CEs to orchestrate any possible diagnostics to be run on the global hardware, the initialization of global memory, synchronization of the other clusters in the configuration, loading of the kernel, and resetting of the cluster back to a state necessary for Xylem to run. If this startup code determines that there are any problems while trying to initialize and configure the system for Xylem, it uses an interactive mode to prompt the operator for help in bringing the system up. During interactive mode, the operator may complete the boot or start all over again to boot the system using the same or a modified system configuration.

The system configuration is represented in a file that the Xylem startup code reads when it begins execution. This file specifies, among other things, what physical memory hardware should be used for global memory, whether this cluster is the master of this configuration, and a physical memory map for global memory.

By specifying the physical memory hardware, the startup code can force Xylem to simulate global memory in a portion of cluster memory specified by the physical memory map. Also, carefully arranged configuration files on several clusters make it possible for several disjoint instances of Xylem to be running using non-overlapping portions of global memory. In this manner, for example, the operating systems group could have two clusters of Cedar running a test version of Xylem while users are running applications on two other clusters with little fear of interaction or side-effects between the two.

The need for one cluster being a master of the configuration has to do with global memory initialization and kernel loading. Only one cluster should initialize global memory and all the other clusters should not access global memory until the master has completed the initialization. Also, the master cluster loads the kernel from disk into global memory and its own cluster memory. Its cluster memory copy is patched to identify itself as cluster 1 of the configuration. Non-master clusters load the kernel from global memory into their cluster memory and patch their copy with a unique cluster identifier.

1.3 Exception Handling (P. Emrath)

Specifications for new exception handling software have been drawn up for Xylem. One aspect of this work is to divide the standard set of UNIX signals into those which are "true" signals, i.e., interrupts by nature, and those which are processor traps, such as addressing errors and arithmetic exceptions. New traps are defined for various new types of addressing faults related to the Cedar architecture. This work also includes specifications for an intertask (or "cross-cluster") interrupt facility. These facilities will be needed by Cedar Fortran in order to implement "shared cluster" memory.

2. OPERATING SYSTEM RESEARCH

2.1 Multiprocessor Run-Time Scheduling (A. Ray Miller)

Research was concluded on the run-time scheduling model under investigation and has been described in [Mill87]. Increases in high speed hardware have mandated studies in software techniques to exploit the parallel capabilities. The effects a run-time scheduler has on a multiprocessor have been examined. The model consists of directed, acyclic graphs, generated from serial Fortran benchmark programs by the parallel compiler Parafrase. A multitasked, multiprogrammed environment is created. Dependencies are generated by the compiler. Tasks are bi-dimensional, i.e., they may specify both time and processor requests. Processor requests may be folded into execution time by the scheduler. The graphs may arrive at arbitrary time intervals. The general case is NP-hard, thus, a variety of heuristics are examined by a simulator. Multiprogramming demonstrates a greater need for a run-time scheduler than does monoprogramming for a variety of reasons, e.g., greater stress on the processors, a larger number of independent control paths, more variety in the task parameters, etc. The dynamic critical path series of algorithms performed well. Dynamic critical volume did not add much. Unfortunately, dynamic critical path maximizes turnaround time as well as throughput. Two schedulers which balance throughput and turnaround time were studied. The first requires classification of jobs by type; the second requires selection of a ratio value which is dependent upon system parameters.

2.2 Mass Storage Systems (G. Heuring)

A network simulator for file system hierarchies has been developed and verified for a system with a small number of clients or workstations. Current efforts are being expended on expanding the simulator to operate on larger numbers of client servers. The simulators are being driven from file traces taken from various computer systems.

Work is still being done to verify the observation that on certain databases Smith's StochOpt file migration policy using file size groups does worse than the Space Time Working Set migration policy.

2.3 Index (P. Emrath, D. Wright)

The 2nd half of the INDEXing programs, which searches text for a specified list of index subjects, will be ready for limited distribution by April 15. People willing to create their own subject lists will be able to run this phase of the program. It provides a reliable way to insure that index entries have been accurately indexed to the proper pages (subject, of course, to further testing against more obscure troff constructs).

The 1st half of the INDEXing programs, which is supposed to provide assistance creating a subject list, could be improved. It can currently be adapted to mark potential entries throughout the text, but the

algorithm that is being used to automatically select potential index entries still needs work. It may be impractical to improve upon the idea of letting the user scan through his text, marking potential subjects, if the algorithm cannot be made more reliable.

2.4 Synchronization (D. Jayasimha and M. Loui: Dept. of Electrical and Computer Engineering)

An analytical study of different schemes of parallel access to synchronization variables was done in [Jaya87]. A simulator (which is a modification of an existing simulator due to N. F. Tzeng) to study these schemes is being implemented and tested. The input to the simulator is an address trace which reflects the effects of finite size caches, and data and instruction pre-fetching [LeYL87a,b].

2.5 I/O Performance Enhancement (P. Emrath, T. McDaniel, D. Padua)

An examination of program-level I/O behavior has begun. A library of routines which trace the I/O calls executed by a program has been written. These routines record the I/O routine name, its arguments, the number of bytes transferred, and a timestamp. This timestamp is the total CPU time of the program, minus an adjustment for the tracing routine itself. Trace data for certain system programs have been collected; and programs have been developed for plotting the resulting trace data. The goal of this work is to evaluate the I/O performance of existing programs, to suggest ways to improve their performance, and to estimate or measure the improvement.

3. CEDAR FORTRAN

3.1 Cedar Fortran (J. Hoeflinger)

The first Cedar Fortran program of significant size was written and debugged. It does a parallel search for the solution to a Rubik's Cube. Without turning on optimization of the Fortran code, a speedup of 3 on a 7-processor Alliant was achieved over the same program run sequentially. Unfortunately, a bug in the Alliant Fortran compiler prevents optimization of that particular program.

The transformation for the SDOALL loop (a parallel DO loop whose iterations are "spread" across Cedar clusters) has been completed. The only limitation to its use is that two SDOALL loops cannot be nested, although a single SDOALL loop, and any number of CDOALLs and sequential DO loops can be nested in any fashion. Future development may remove the restriction on nesting SDOALLs.

A QUIT statement has been added to both CDOALL and SDOALL loops. This statement terminates one or more parallel loops. There is an optional label which can be used on the QUIT statement. If it is present, execution proceeds at the statement with that label after the loop (or loops) terminates. The label determines which loops actually terminate. Conceptually, jumping out of the scope of a parallel loop terminates that loop. Because of this, a single QUIT statement may cause termination of several parallel loops by specifying a label outside the scope of all of them.

A LOOP statement has also been added to define a "preamble" for either a CDOALL or SDOALL loop. That preamble is executed once on each processor prior to the first iteration which the processor performs. The preamble of the loop is defined to be all of the statements between the SDOALL or CDOALL statement and the LOOP statement.

The transformation for DATA and SAVE statements (as well as initialization within a type declaration) is nearing completion. Conceptually, all initialized variables in the same storage class (shared global, private cluster, etc) are placed in a COMMON statement of that storage class and initialized in a separate BLOCKDATA program unit.

An existing program ("txo") was fixed to properly translate a Concentrix object file to a Xylem object file. The major problem faced by this program is that the Alliant Fortran compiler does not make separate sections for COMMON blocks, but rather places all COMMON blocks within a single section named "_DATA_". The "txo" program now correctly extracts the COMMON blocks, making separate sections for each, and fixes up all relocation records correctly.

Support for vector assignment within a FORALL statement was also added.

3.2 Cedar Fortran Programmer's Manual (M. Guzzi)

The first version of the *Cedar Fortran Programmer's Manual* has been revised. There are now two versions of this document, a complete version that contains confidential material [Guzz87] and a public version [Guzz86]. These documents describe multitasking concepts, all language extensions, and library routines in Cedar Fortran. The first official releases of these documents will be presented during the Cedar Fortran Review.

3.3 Runtime Libraries (M. Guzzi)

The Alliant Fortran and C runtime libraries have been made to work with Cedar Fortran. Cedar Fortran with multitasking and synchronization routines has been implemented. Work continues on microtasking routines.

3.4 Processor Self-scheduling and Microtasking (M. Guzzi)

Use of microtasking in Cedar Fortran has been restricted to the scheduling of SDOALL loops. No further development of microtasking will be done until the Cedar hardware becomes available for experimentation. SDOALL loop implementations are being incorporated into the Cedar Fortran Runtime system.

3.5 Kap (M. Guzzi)

Investigation began on modifying Kap to accept Cedar Fortran. The use of comment directives to achieve the functionality of Cedar Fortran while allowing the development of portable code is being investigated.

3.8 Compiler Development (S. Midkiff)

Programs to assist in the creation of DSEE models for new versions of KAP were completed. A new version of KAP was received and ported, using the new versions. The new KAP changed the way that vector Fortran was generated; this in turn broke the software written in the last quarter of 1988, and the first weeks of 1987. Efforts are now being directed towards understanding what the problem is, and fixing it.

4. SYMBOLIC PROCESSING

4.1 Lisp (W. Harrison, D. Padua)

Work continues on the Parcel compiler. Most of the scalar (non-parallelizing) optimizations are completed, and the compiler has been augmented to provide for a simple form of separate compilation. This is one of several measures which have been taken to facilitate the ultimate self-compilation of the compiler. Passes which have been completed to date include interprocedural value propagation, data dependence testing and side-effect detection, clean variable marking (used to detect expressions over quantities which are unaltered following the formation of a closure; such an expression can be lifted out of the closure, and made part of the lexically surrounding function), self-recursion detection, series-parallel normalization (used to produce a highly regular control flow graph, which consists of nested single entry regions), binding allocation (used to decide, for each lambda expression in the program, whether the variable bindings of that lambda expression are to be allocated on the heap or on the stack), dead code elimination, and constant folding.

More work has been done in advancing the accuracy of the compiler's interprocedural data dependence testing. In particular, a test has been incorporated into the routines which perform interprocedural value propagation to permit the identification of data structures which are trees, that is, data structures such that any two instances of a mutable object which are found by following distinct paths from the root of the structure are necessarily distinct objects (occupy distinct memory locations). This is most helpful,

as it facilitates the parallelization of a traversal of the structure, even if that traversal makes modifications to the mutable objects it encounters. This technique appears to be directly applicable to structures of records as might be constructed by a Pascal or C program, as well.

The coding of the (function-level) parallelizing transformations has begun. These transformations include the primary transformations of recursion splitting and exit-loop translation.

4.2 Parallel Prolog Interpreter (D. Sehr and L. Kale:Dept. of Computer Science)

Work continued on the development of the model for the Or-parallel evaluation of Prolog programs containing side-effects. Several errors were corrected in the original model. The algorithms were re-coded using monitors which should be easier to verify. Work has begun on implementing the model in an interpreter on the Alliant FX/8, and will continue through the Summer. This interpreter will employ the Edinburgh syntax, and will support the standard set of built-in predicates found in sequential interpreters. It is hoped that this interpreter will serve as a suitable test facility for binding environment evaluation, as well as demonstrating that Prolog can be executed in Or-parallel while allowing side-effects.

5. COMPILER RESEARCH

5.1 Data Location (S. Karlovsky, D. Padua)

The trace-driven shared memory multiprocessor simulator has been designed and is currently in development. The simulator accepts the output from the previously developed trace generation pass at the back end of the Cedar Fortran Preprocessor. These traces include the memory reference, interprocessor synchronization, and concurrent loop startup and cleanup information necessary to facilitate the simulation of a shared memory multiprocessor with an arbitrary number of computational elements. Initial experiments to be performed using the simulator are expected to focus on the problem of maintaining cache coherence in a multicache system.

5.2 Debugging (T. Allen, D. Padua)

A simple prototype debugger is being brought up. The prototype debugger will have roughly the same functionality as the more sophisticated design presented in [AlPa87]. However, due to efficiency constraints, it will be useful for a much smaller class of problems. The prototype will be available for immediate testing and will provide for an opportunity to evaluate this approach to debugging. By actually using the debugger during the development of real programs, it is hoped that insights may be gained that should be incorporated into a production debugging tool.

5.3 Symbolic Algebra (D. Padua, P. Petersen)

The first two steps have been completed in porting the MACTRAN⁴ system to REDUCE, the display of indexed variables, and the discretization of a PDE from the continuous domain to the discrete domain. A parallel effort is being made to port the MACTRAN system from Mac-Lisp MACSYMA to Franz Lisp MACSYMA under UNIX. This port will allow development to proceed in the best suited environment. The parts of the port completed are: display of indexed variables; discretization of PDEs; linearization of PDEs; and coefficient generation using alternating direction implicit method (ADI). Work is underway on the macro command translator and template library for Fortran code generation from the MACSYMA equations.

5.4 Functional Programming Language (S. Ghosh)

During this quarter, techniques to improve programs were investigated and a functional programming language was chosen.

⁴Wirth, M. C., *On the Automation of Computational Physics*, Ph.D. Thesis, Univ. of California, Davis, 1980.

The program improving techniques were principally techniques to either remove recursion completely from certain programs, or to improve the efficiency of some recursive programs. Similar techniques will be used to improve programs coded in our functional programming language.

The other effort was directed at choosing a functional programming language. The intention is to implement a parallel functional language on the Cedar machine, and to devise new techniques to parallelize functional programs. Several common functional languages for this were investigated.

In the following quarter, efforts will be directed at making progress in implementing the parallel functional language and devising program improvement schemes in that language.

5.5 Parafrase II (M. Girkar, B. Leung, C. Polychronopoulos, D. Schouten)

The design phase of Parafrase II is near completion and the implementation phase has started. It was decided that the goal of allowing many languages as input would be accomplished by having an intermediate representation which would be acted on by the core of Parafrase II. The output from Parafrase II will be the same intermediate representation. Preprocessors and postprocessors will reproduce the original source for each language augmented with parallel constructs, from the intermediate representation. Work on the C preprocessor has started.

6. SCIENTIFIC WORKSTATION ENVIRONMENT: FAUST

6.1 Faust Editor (Y. Gaur, V. Guarna, D. Jablonowski)

Functional and design specifications have been written, and some preliminary review meetings have been held to discuss both the PAS System and the Graph Manager of Faust. The PAS (Program Analyzer/Synthesizer) System is an application of Faust used to examine a program at abstract levels, and to construct a program from abstract levels. Design efforts have focused on the development of a package of utilities and abstract data types that will be used by the integrated Faust tools.

Version 10, release 4 of the X Window System has been ported to the local Apollo ring. X windows will form the foundation for all user interface functions controlled by Faust. Preliminary specifications for the Faust I/O support utilities have been developed.

A preliminary lexical scanning tool was developed that will generate the necessary static call graph information that is present in arbitrary C programs. This tool will later be enhanced to recognize Cedar Fortran and will be used by the Faust system to build the Subroutine Interconnection Graph when requested by the user.

Apollo's Network Computing System tools have been brought in-house for experimentation. This package includes utilities to allow remote procedure call services to be supported across a wide variety of machines. Experiments to test the performance and portability of these tools are being developed.

6.2 Faust Debugger (J. Krause, P. Lewis, D. Reed)

The debugger metalanguage specification has been completed. A prototype debugger metalanguage parser was developed and is running on the Alliants. The source code for Alliant's DBX has been acquired and is being analyzed to see how it should be modified to work in the Faust environment.

6.3 Faust Expert System (M. Berry, S. Bhansali, M. Harandi)

The existing expert system generator (GPSI) has been successfully ported to the Apollo system. Preliminary work toward building the knowledge base pertinent to numerical library routine selection has begun. Additionally, enhancements to the GPSI system have been laid out (and implementation begun). These enhancements include a new user interface style (multiple facts with one query), a new decision tree node that will allow the invoking of external functions, and a more comprehensive trace-back feature. In preparation for a future AFOSR site visit, demonstration rule bases from application areas such as structural dynamics and computational fluid dynamics are being designed.

6.4 Parallel Ray Tracing (D. Defend, D. Gannon, A. Tuchman)

Research has taken a different direction than indicated in the previous progress report. The plan was to experiment with scalar concurrency and vector parallelism as implemented on the simple ray tracer (Basic Ray Tracing Package by Knauss) and on a more sophisticated ray tracer (Tracer by Ray Tracing Corporation). Because of other interests and the unavailability of software for the new ray tracer, research has been directed at further experimentation with the simple ray tracer.

Work is currently being done to modify the current ray tracing algorithm to make the ray tracing process more efficient, seeking modifications which enhance scalar and vector parallelism. The main focus of attention is on a ray tracing technique⁵ sometimes referred to as *space tracing*. This involves the addition of a preprocessing step to the tracer which divides the scene into volume elements, or *voxels*, and determines which of the objects in the scene overlap each voxel. With this information, it is possible to make the sphere intersection process more efficient because the tests can be localized for a particular region of space. The work being done includes the implementation of space tracing into the current ray tracing algorithm. Then various approaches to exploit vector and scalar parallelism are experimented with in the new parts of the algorithm.

6.5 Automatic Error Analysis and Faust (B. Bliss, E. Gallopoulos)

Work is under way for the integration of the automatic error analysis package⁶ ⁷in the Faust environment. The front-end of the original package is based on Miller's mini-compiler⁸ which can only handle programs written in a simple Fortran-like language. Our current work is designed to replace this front-end with an interface which will be able to handle Cedar Fortran programs.

⁵Kaplan, M. R., *Space Tracing, a Constant Time Ray-Tracer*, SIGGRAPH85 Conference Proceedings, July 1985.

⁶Larson, J. L. and A. H. Sameh, *Algorithms for roundoff error analysis - a relative error approach*, Computing, v. 24, pp. 275-297, (1980).

⁷Larson, J. L., M. E. Pasternak, and J. A. Wisniewski, *Algorithm 594: Software for relative error analysis*, ACM TOMS, v. 9, pp. 125-130, (March 1983)

⁸Miller, W. and D. Spooner, *Algorithm 592 - Software for roundoff analysis [2]*, ACM TOMS, v. 4, pp. 388-390, (December 1978).

APPLICATIONS QUARTERLY PROGRESS REPORT FIRST QUARTER 1987

1. NUMERICAL LIBRARY

1.1 Solving Dense and Banded Linear Systems

1.1.1 Iterative Methods for General Sparse Linear Systems (E. Anderson, Y. Saad)

A parallel algorithm to compute the incomplete LU decomposition of a sparse matrix has been written. Given the nonzeros and the row structure of a sparse matrix, the new algorithm forms the column structure so that both row and column accesses can be done efficiently. A previous version without the additional structure could only combine previously determined rows to the current row in each elimination step. The new version can combine the current row with later rows. While the former algorithm was almost totally sequential, the latter shows a nearly linear increase in performance with the number of processors.

Some comparisons of the ILU algorithms have been made. The sequential algorithm showed a speedup for 8 processors vs. 1 processor of at most about 1.05, while the parallel algorithm shows a speedup of about 3.5 to 4.0 in a typical case. Because of the overhead involved in computing and maintaining the column structure, the parallel algorithm tends to run slower for a level of fill-in of 0, about the same for level of fill-in 1, and up to two times faster for higher levels of fill-in.

Two aspects of this implementation of the ILU decomposition will be the subject of further study. The first is the fixed storage scheme. When overflow occurs within a given row, the smallest entry in absolute value is deleted. Limiting the size of the preconditioning matrix will decrease the time for the iterative method but may harm the convergence properties. The second is the column structure, which will allow partial pivoting to be added at little additional expense. Pivoting by rows will increase the number of problems to which the ILU preconditioning can be applied.

1.1.2 BLAS3 Primitives (K. Gallivan, W. Jalby, U. Meier)

This quarter a set of BLAS3 primitives which is a superset of the Cray numerical library routines were developed. A summary document of the features and performance of the routines is in preparation. Typical performance of a matrix multiplication with these routines is approximately 50 Mflops.

1.1.3 High-level Modules (K. Gallivan, W. Jalby, U. Meier)

The paper *The Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design*, CSRD report 625 [GJMS86] has been completed. A slightly altered version will be submitted to IEEE Transactions on Computers. The final versions of the codes used for this paper achieved: 52 Mflops for dense matrix multiplication; 50 Mflops for a rank-64 update; and 43-45 Mflops for both a block LU decomposition and a block Modified Gram-Schmidt.

The pipelined Given's transformation code developed earlier has been extended into a block version. This code is presently being used in an investigation into algorithms for Kalman filtering.

1.1.4 Direct Schemes for Solving Block-Tridiagonal Systems (M. Berry, A. Sameh)

Implementations of Complete and Incomplete Block Cyclic Reduction for solving block tridiagonal diagonally dominant linear systems on the Alliant FX/8 has been completed. Preliminary experiments for block sizes less than 16 indicate the Complete Block Cyclic Reduction (CBCR) can achieve a speedup of 2 over both DGBFA+DGBSL from LINPACK and the previously implemented *Spike* solver scheme for small block sizes. In taking full advantage of the degree of diagonal dominance, Incomplete Block Cyclic Reduction (IBCR) can demonstrate speedups as high as 3 over the LINPACK routines. The algorithm complexity and associated redundancy factors of BCR and IBCR are significantly less than that of the initially proposed *Spike* solver for small block sizes. Given the large overhead incurred by the current *Spike* scheme, a

new implementation in the nature of *recursive doubling* is now being developed to improve the performance of the Spike scheme relative to BCR and IBCR. Regarding speedups for parallelism, BCR and IBCR achieve ratios near 5 while the Spike scheme with a redundancy factor of approximately 3 can achieve ratios of 6 (block sizes ≤ 16).

For block sizes ≥ 32 an implementation of Block Gaussian Elimination (BGE) using Block LU factorizations has achieved speedups near 2 over DGBFA+DGBSL from LINPACK and speedups near 3 over the Spike Scheme. The success of BGE for the larger block sizes can be attributed to the superior performance of the block LU algorithm for diagonally dominant matrices. A factorization scheme is now under development to produce a factorization for block tridiagonal matrices which can be used repeatedly for various number of right hand sides. A similar scheme using block Cholesky factorizations for the symmetric positive definite block tridiagonal case is being developed for structural dynamics applications. Implementations of all the schemes discussed above on the Cray X-MP/48 are planned, with some results expected before the end of the second quarter. A CSR report detailing this research effort should be available at the time in preparation for the presentation of a paper entitled *Multiprocessor Schemes for Solving Block Tridiagonal Linear Systems* at the Paris ICIAM Conference in July.

1.1.5 An Iterative Scheme for Solving Block Tridiagonal Systems (R. Bramley)

The solution of block tridiagonal systems by means of projection methods is being investigated. Such a system is first permuted to form two interconnected systems, each with an independent block structure suitable for concurrency [KaSa86]. The set of solutions to each subsystem forms an affine set, and the overall solution is sought as the intersection point of those two affine sets.

Orthogonal projections related to the subsystems are well-understood but expensive to calculate, requiring the solution of least-squares subproblems. Testing has shown that methods based on them can require an inordinate number of iterations if the affine sets are separated by small angles. However, some oblique projections can be much cheaper to calculate, requiring only matrix-vector multiplications. The theoretical background for using oblique projections is not well developed, therefore, current work is concentrated on establishing such a background.

1.1.6 Narrow Banded Solvers (C. Ulrich, W. Harrod)

A general parallel algorithm for solving narrow banded systems directly, factoring narrow banded systems, and solving narrow banded triangular systems directly has been investigated.

Parallel solvers for the special cases of lower 3-diagonal and 2-diagonal systems, as well as tridiagonal and pentadiagonal systems were implemented along with the parallel LU decomposition of tridiagonal and symmetric tridiagonal matrices. In all cases, stability appears to be assured if the original systems are column diagonally dominant.

The lower triangular linear systems have been optimized by using loop-unrolling and load-balancing across the processors. The performance of the tridiagonal and pentadiagonal implementations is currently being enhanced in similar ways. Current performance figures are as follows:

Speedups are sequential over parallel for $n = 1000$ unknowns	
unit lower 2-diagonal	5.3
unit lower 3-diagonal	4.1
tridiagonal solver	4.0
pentadiagonal solver	3.0
tridiagonal LU decomposition	3.3
symmetric tridiagonal LU decomposition	2.8

These solvers [Uri87] have been used in developing a parallel QR algorithm for the tridiagonal eigenvalue

problem⁹.

1.1.7 Block Reduction to Tridiagonal and Hessenberg Forms (W. Harrod)

The first step in the algorithm for computing the eigenvalues of a real general matrix is to reduce the matrix to upper Hessenberg form by similarity transformations. A block scheme has been designed for this reduction process either via block Householder transformations or block elementary transformations. Reduction by orthogonal transformations has been implemented. The block algorithm is 6 – 8 times faster than the EISPACK subroutine orthes. A similar code for the orthogonal reduction of symmetric matrices to the tridiagonal form is currently being debugged.

1.2 Eigenvalue Problem

1.2.1 Singular Value Decomposition (SVD) (M. Berry, A. Sameh)

Performance experiments for HQRJAC, the technique for determining the singular value decomposition of real rectangular matrices by combining the Block Householder QR factorization and One-side Jacobi scheme, are still underway. The application of the Householder vectors to produce the appropriate left singular vectors has been implemented and tested.

1.2.2 Tridiagonal Eigenvalue Problem (C. Baucom, D. Sorensen)

Work has continued on modification of existing Tridiagonal Eigenvalue Problem code based on rank one tearing, which uses the SCHEDULE package, [DoSa86]. A more space-efficient approach is being implemented in which it is not necessary to store the array of new eigenvectors found at each level of the divide and conquer scheme.

1.2.3 Trace Minimization Algorithm for the Generalized Eigenvalue Problem (W. Harrod, A. Sameh)

The problem of approximating a few of the smallest (or largest) eigenvalues of $Ax = \lambda Bx$, where A and B are large, symmetric, and sparse, with B being positive definite is under investigation. A "trace minimization" algorithm with shifting strategy has been implemented on the Alliant FX/8. This method uses routines for computing the eigenvalues and corresponding eigenvectors of a small dense matrix, the orthogonal factorization of a matrix using a block Modified Gram-Schmidt algorithm, and a block conjugate gradient algorithm for linear systems with many right hand side vectors. Implementation of an improved convergence acceleration strategy and comparisons with the Lanczos algorithm are underway.

1.3 Conjugate Gradient Algorithms (W. Jalby, U. Meier, A. Sameh)

The test codes, that were used in the second and third quarter of 1986 to test several conjugate gradient algorithms for symmetric positive definite linear systems arising from five-point difference schemes for elliptic differential equations, were reformulated into library quality routines in the first quarter of 1987. The algorithms coded comprise the classical conjugate gradient algorithm (CG) and two preconditioned versions, a vectorized version of the incomplete Cholesky preconditioner (VINVCG) and a polynomial preconditioner (POLCG).

1.4 Nonlinear Optimization and the Solution of Nonlinear Equations

1.4.1 Nonlinear Equation Solver (R. Guo, Y. Saad)

Work is continuing on implementing numerical methods for solving general large nonlinear systems of equations. Currently, experiments with homotopy strategies are being conducted to determine their effectiveness in improving the global convergence properties of the nonlinear GMRES technique.

⁹Sameh, A. and D. Kuck, *A Parallel QR Algorithm for Symmetric Tridiagonal Matrices*, IEEE Transactions on Computers, vol. c-28, no.2, Feb. 1977, pp. 147-153.

1.4.2 Numerical Conformal Mapping Method (M.-Q. Chen, A. Sameh)

While optimizing and testing the package¹⁰ of routines for solving the Poisson equation on irregular domains by a numerical conformal mapping method adapted from the Chakravarthy-Anderson¹¹ method, other numerical conformal mapping methods are being reviewed. The study mainly focuses on the complexity of the methods and the potential for parallel computation.

1.4.3 Partially Separable Large-scale Optimization (M.-Q. Chen, S.-P. Han)

A parallel algorithm for solving partially separable minimization problems, as mentioned in the last QPR, by using the updated conjugate subspaces method¹² with the partitioned updating technique¹³ is described in [ChHa87]. The program which implements this algorithm has been completed and tested for several standard problems with a partially separable structure on the Alliant FX/8. The achievement of average speedups of 1.91, 3.29 and 5.27 on 2, 4 and 8 CEs, respectively, indicates that the algorithm is suitable for parallel computation. Speedup on i CE's is defined as
$$\frac{\text{CPU time for using 1 CE}}{\text{CPU time for using } i \text{ CEs}}.$$

For problems of the form:

$$\min_x f(x) = \sum_{i=1}^m f_i(x), \quad x \in R^n,$$

the storage required by this algorithm requires m pairs of matrices, each pair i being of dimension n_i , for $i=1, \dots, m$, where n_i is the number of nonzero components of x of the element function f_i . When $n_i \ll n$, for $i=1, \dots, m$, this algorithm requires little storage.

1.5 Rapid Elliptic Solvers (E. Gallopoulos, Y. Saad)

Work focused on the block cyclic reduction (BCR) based methods. The initial results, as well as the technique used, are described in detail in [GaSa87]. A new method has been developed which is used to overcome this sequential bottleneck and in turn introduce as much parallelism as needed at each step of the computation. The approach can be used successfully on a majority of the new supercomputers. An important advantage of the method is that it allows the efficient parallelization/vectorization of NCAR's FISHPACK¹⁴ which is entirely based on BCR for two-dimensional problems. Hence, one can solve problems more complicated than Poisson's equation with no restrictions on the resolution or the boundary conditions. Moreover, it can be used successfully on computer systems on which the FFT algorithm is expensive. With little modification to the original (FISHPACK) code, application of the method to a simple Poisson problem on a 255×255 grid in a square region with Dirichlet boundary conditions results in it running approximately 4 times faster than the original code on the Alliant FX/8.

1.6 Kalman Filtering (K. Gallivan)

An effort was started this quarter to investigate the implementation on the Alliant of a Kalman

¹⁰Kamgnia, E. R., *A Numerical Conformal Mapping Method and Poisson Equation on Irregular Domains*, report no. UIUCDCS-R-83-1157, University of Illinois, Urbana.

¹¹Chakravarthy, S. and D. Anderson, *Numerical Conformal Mapping*, Math. Comp.R, 33(1979) pp. 953-969.

¹²Han, S.-P., *Optimization by Updated Conjugate Subspaces*, 1985 Dundee Biennial Conf. on Numerical Analysis, Univ. of Cambridge, England.

¹³Griewank A. and Ph.L. Toint, *Partitioned Variable Metric Updates for Large Structured Optimization Problems*, Numerische Mathematik, 39, 119-137 (1982).

¹⁴Swartrauber, P. and R. Sweet, *Efficient Fortran Subroutines for the Solution of Elliptic Partial Differential Equations*, NCAR Technical Note, IA-109, Boulder, July, 1975.

filtering algorithm¹⁵. The method uses the fact that the Kalman filtering problem can be treated as sparse recursive least squares problem. Methods based on plane rotations, Householder reflectors and the Modified Gram-Schmidt algorithm will be considered. At present, three codes have been implemented: two versions for smaller problems, filters of degree less than ninety-six, based on the pipelined Given's rotation code developed earlier; and one based on an initial implementation of a block Given's rotation code developed this quarter. This work also involves some interaction with Dr. K. Tait of TASC who is working on an Alliant implementation of a similar method.

1.7 Strategies for Cache and Local Memory Management by Global Program Transformation (K. Gallivan, D. Gannon, W. Jalby)

An investigation has started into developing strategies by which a compiler could analyze the data dependencies of piece of code and automatically generate a cache control policy which would efficiently exploit a hierarchical memory system. A paper on this topic will be presented at the International Conference on Supercomputing 1987.

2. APPLICATIONS

2.1 Circuit Simulation (K. Gallivan, Y.-C. Ju, P. Koss, S. Lo, R. Saleh, D. Sorensen, X. Wang, E. Xia, G.-C. Yang)

2.1.1 Nonlinear and Waveform Relaxation

The sequential Waveform-Newton code, SPLAX, from U. C. Berkeley is now available on the Alliant. A preliminary version of a parallel implementation of the algorithm has also been completed. A paper discussing this method and the considerations involved in developing a parallel version will be presented at ICCD 1987 (Saleh and Xia).

The sequential code RELAX2.3 has been successfully converted into a parallel waveform relaxation code which allows a Gauss-Jacobi iteration or a statically scheduled Gauss-Seidel iteration which exploits the parallelism implicit in the partial ordering of the subcircuits. A Gauss-Seidel iteration which uses dynamic scheduling is under development.

Codes which allow the assembling of the Jacobian in parallel have been integrated into RELAX2.3. One version uses a simple single-lock strategy and a second uses a multiple-lock strategy. Both, however, suffer some degradation due to the use of locks around critical sections of the code. A third version which does not require locking of critical sections of code has been designed. In this approach, all of the contributions to the Jacobian of the various devices are computed in parallel and written to individually private locations. After a single barrier synchronization point, each element of the Jacobian is considered in parallel and the contributions are added together to form the final Jacobian. This strategy is applicable on processors where the degree of parallelism available is limited compared to the number of devices in the circuit. The second part of the computation is trivially vectorizable. Vectorization of the computation of the contributions is, however, more complicated and under consideration.

An improved version of PSPLICE is now available on the Alliant. This version differs from the first in its implementation of dynamic scheduling. At present, the second version is undergoing performance tuning.

Work has begun on a hybrid waveform/nonlinear relaxation code which will allow hierarchical processing of circuits on the Cedar machine.

¹⁵Paige, C. and M. Saunders *Least Squares Estimation of Discrete Linear Dynamic Systems Using Orthogonal Transformations*, *Siam Journal of Numerical Analysis*, April 1977

2.1.2 Direct Methods

During this quarter, further enhancement of the parallel SPICE code was completed and more impressive results were achieved. An average speedup of approximately 6.5 has been achieved for the modules LOAD and LTE. This represents a speedup with respect to the total time for the code of about 3.4. Both of these figures represent speedups of the parallel version on one and eight CE's. The parallel code on a single CE incurs approximately fifty percent overhead relative to the sequential version of SPICE. The current parallel version of SPICE, however, achieves a speedup of twenty-two relative to a Vax 11/780, which is about the same as the Cray-1S.

The enhanced performance achieved this quarter was due to the fact that the number of statements in the critical region has been reduced by a factor of ten. A new effort has been started during this quarter to design a SPICE version of the lockless scheme, mentioned above, which does not require the critical region. Such lockless scheme seems to require a fundamental change to the data structures and is much more complex than the approach currently used in SPICE.

2.2 Computational Chemistry (A. Galick, W. Harrod)

Work continued on restructuring R. Wyatt's program for computing quantum mechanical time-dependent transition probabilities.

Actually, two versions of this program have been developed. The first is a Fortran-77 (maybe even Fortran-86) benchmark which is portable to at least the Vax. The second is an Alliant FX/Fortran version which runs only on the Alliant machines. Unlike the original code, in which a separate program ("photon") generated the input data, these versions both generate input data in a subroutine. Incorporated directly as a subroutine, "photon" was the most expensive subroutine. After modification, a speedup of approximately 7 was achieved in "photon".

2.3 Computational Fluid Dynamics

2.3.1 Computational Fluid Dynamics (R. Roloff, B. Semeraro)

The purpose of the computational fluid dynamics activity in this quarter has been to adapt CFD software to parallel computers. The algorithms have not changed yet; but the software is currently being modified for the FX/8.

Computationally intensive kernels have been extracted from CFD codes. These kernels have been modified to improve parallel efficiency on the Alliant FX/8. Three codes are being analyzed and are discussed here in order of their increasing complexity. Memory bandwidth limitations will dictate changes to these kernels if one is to achieve high parallel efficiency.

The Nascrin code by A. Kumar at NASA Langley has the simplest algorithm, a MacCormack explicit-implicit scheme. The calculations here, vector adds and vector multiplies, achieved 66% efficiency on the Alliant FX/8.

The Arc3d code by T. Pulliam at NASA Ames uses a Beam-Warming algorithm and achieves an estimated 75% parallel efficiency. The Beam-Warming algorithm requires solution of block-tridiagonal linear systems.

The code with the highest parallel efficiency is a spectral method code for turbulent channel flow by J. McLaughlin at Clarkson University. Because the processors compute trigonometric functions slowly, the bandwidth needed to keep the processors supplied with data is less than either of the previous algorithms. This spectral code achieves an 87% efficiency.

We are producing pressure contours using color scales on our Sun 110's. The NCAR graphics package has been ported to the Suns by Dave Semeraro. Graphics codes used by Dr. Thomas Zang at ICASE and NASA-Langley have been obtained. These will be used for visualizing the turbulent flow results obtained using Zang's channel code. They will be of use for developing similar graphics codes on the Pixar.

2.3.2 Convection-Diffusion P.D.E. Solver (J. Scroggs, D. Sorensen and R. Chin and G. Hedstrom: Lawrence Livermore National Laboratory)

An accuracy test was added to the convection-diffusion code. Computation of the residual demonstrated that the accuracy is (10^{-3}), the magnitude of Δz . This confirms the theoretical error estimate. We discovered that it was necessary to compute the residual in the local coordinate system, due to the ill-conditioning of the problem in the physical coordinates.

Memory locality, or lack thereof, was ruled out as a cause of the poor efficiency on the Alliant FX/8 (the poor efficiency was reported in the second QPR of 1986). Partitioning memory for one of routines demonstrated a variation in run time of .05 seconds out of a total execution time of 4.5 seconds for the routine. Furthermore, by removing most of the Fortran IF statements used for debugging the program, a speedup of about 6.3 was obtained (Efficiency of about .8). Thus, the problems causing the poor results obtained last quarter were perhaps due to a bottleneck process executing several inefficient IF statements.

A paper entitled "Parallel Computation of Domain Decomposition Method" [CHSS87] was written for the IMACS '87 conference. This paper summarizes the work completed thus far on the linear problem, and presents a novel shock detection scheme discovered by R. Chin, to be used in the nonlinear problem.

2.3.3 Navier-Stokes Solver (W. Jalby, U. Meier and L. Hageman: Westinghouse)

Work is continuing on the implementation of a software package for solving the two-dimensional time-dependent Navier-Stokes equations for an incompressible fluid on a multivector processor. A visit to Lou Hageman was made at the end of February to discuss the design of the package and other open questions. Preliminary documentation of the package including a flowchart and a description of the subroutines and their parameter lists has been finalized and coding and testing of some parts of the code are completed.

2.4 Structural Mechanics and Dynamics

2.4.1 Plate Bending (H.-C. Chen)

Two types of preconditioning techniques were examined for the linear systems resulting from the three simply supported plate bending problems mentioned in the last QPR. The first is the polynomial preconditioning using Jacobi weights and the second the diagonal (or block diagonal) preconditioning. The polynomial preconditioning, up to degree three, did not enhance the performance although the numbers of iterations were indeed reduced. For the second type of preconditioning, both the main diagonal and the block diagonal with blocks of size three were good preconditioners. In all cases, the performance of the SAS approach was always better than that of the non-SAS one.

Research is continuing on the SAS approach for the generalized eigenvalue problems which arise in the modal analysis of plate bending problems. Both lump mass and consistent mass approaches are considered. The coding of a finite element program by employing the SAS approach for the lump mass case has been completed. The program for the consistent mass case is being tested.

2.5 Weather Simulation (E. Gallopoulos, K.-S. Tai)

In order to better understand the numerical behavior of the NMC spectral model, it was ported to the NCSA Cray X-MP/48 and comparisons of the numerical results for the model between various machines and compilation options have been done. The relative magnitude of the quantities involved, made the data checking much more convenient in physical than in wavenumber space. For only a few time steps of simulation, the representative output data sample had only negligible differences between the Cray X-MP/48 (one processor), the Alliant FX/8 (original code running on one CE) and the same machine with the modified code running on 8 CE's. As expected, the differences in the data sample are much more substantial after 12 hours of simulation (42 time-steps). Little evidence has been found in the literature for the behavior of spectral models on computers with different floating-point representations, rounding techniques, and floating-point arithmetic. Even less evidence exists for vector multiprocessors where, in the interests of vectorization and parallelization, the code is substantially restructured, inner products and

other reduction operations are accumulated differently.

The original code was converted to double precision on the Alliant. Thus, some problems were found in the way double-precision, complex arithmetic is handled on the Alliant's current Fortran compiler. It is also not clear how much is gained with the extra precision in weather modeling.

A new, fully parallel, method was developed for the calculation of the roots of the Legendre polynomials to substitute the original method used with the NMC model in order to find the Gaussian quadrature points and weights. Although it is not expected to affect the runtime of the model (as it is used only once) it may prove useful in other circumstances calling for repeated such calculations.

The tapes containing the NCAR community climate model (CCMOB) which will be used in future experiments have been received.

2.6 Applications Graphics (B. Semeraro, A. Tuchman)

A device simulation code obtained from I. Kizilyalli of the Coordinated Science Laboratory has been ported to run on the Alliant. Minor restructuring of the code was done to obtain better performance. The code is being used as a model for an Alliant-Pixar distributed environment.

Testing of the network communication routines to allow cooperation of processes on different CPU's has been completed. These routines allow Alliant code to send raw data with very high-level graphics instructions to the Pixar. The network interface will be use for sending commands and responses and for synchronization.

NCAR graphics software has been installed on an Alliant. The software will be installed in a commonly accessible library. One of the routines has been modified to allow it to be run as a window-based Sun application.

PERFORMANCE EVALUATION PROGRESS REPORT FIRST QUARTER 1987

1. Performance Evaluation Tools (A. Malony)

Tool development work has been done in all three performance evaluation areas: analytical modeling, simulation and measurement. The Purdue Mean Value Analysis (PMVA) program has been made available on the CSR D Vax. PMVA is a program for analyzing closed product-form queuing network models. A general-purpose, process-oriented simulation system, CSIM, is also now available on the CSR D Vax. Work is underway to port these tools to other CSR D machines.

A study was completed of parallel program profiling in Cedar. [Malo87] details the conceptual problems and the implementation difficulties of this performance evaluation approach. A complete specification of a Cedar profiling tool, *cprof*, based on routine profiling is given. The study concluded, however, that profiling does not provide a versatile enough of a basis for Cedar software performance evaluation tools. A reduced version of *cprof* may still be implemented.

An investigation of software event tracing as a foundation for software performance evaluation tools is underway. This approach subsumes the functionality of the entire class of profiling tools plus provides additional capabilities such as program execution replay. Tracing can also be used for operating system performance evaluation. A specification of a Cedar tracing tool, *ctrace*, is forthcoming.

2. Performance Evaluation (K. Gallivan, A. Malony, W. Jalby, Y. Saad)

The efforts of this group to investigate the effect of processor/memory-system interaction has continued in this quarter. In past quarters the tools of kernel benchmarking and low-level tracing via a logical analyzer have been used to track the influence of the memory system on algorithm performance. In this quarter, an address-stream-driven simulator has been developed which allows the interaction between multiple vector processors and a parallel memory to be modeled in terms of certain parameters. Thus far this simulator has been used to investigate the cache-processor interface of the Cedar cluster. The parameters of interest are: number of processors, number of cache banks, cache bank conflict resolution strategy, cache address mapping, cache size and instruction timings. Preliminary simulations have achieved remarkable results in predicting performance degradation due to the static cache conflict resolution policy used on the Alliant. These predictions have been confirmed via kernel benchmarking.