PRISM: A Laboratory for Research in Future High-Performance Parallel Computing

Computer Science Department
University of Wisconsin - Madison

Technical Report #1016

March 1991

PRISM: A Laboratory for Research in Future High-Performance Parallel Computing

Computer Sciences Department
University of Wisconsin – Madison

March 1991

Project Coordinator: Mary K. Vernon

Additional Principal Investigators:
Michael J. Carey
Charles R. Dyer
Mark D. Hill
Robert R. Meyer
Barton P. Miller

Participating Faculty:
Eric Bach
Anne Condon
Renato De Leone
David J. DeWitt
Michael C. Ferris
Susan Horwitz
Yannis E. Ioannidis
Sangtae Kim
Lawrence H. Landweber
James R. Larus
Olvi L. Mangasarian
Miron Livny
Jeffrey F. Naughton
Seymour Parter
Raghu Ramakrishnan
Thomas W. Reps
Stephen M. Robinson
Jude W. Shavlik
Gurindar S. Sohi
John C. Strikwerda
Prasoon Tiwari
Leonard Uhr
David A. Wood

TABLE OF CONTENTS

A. INTRODUCTION

This report is an excerpt of a grant proposal submitted in September 1990 that has been funded by the National Science Foundation Institutional Infrastructure Program. It describes our plans to develop a state-of-the-art facility for experimental research in parallel computing, and outlines several important research problems that will be addressed using the facility. The budget for the infrastructure includes $2 million from NSF, matched by approximately $1.1M from the University of Wisconsin and DARPA.

The PRISM project will start on June 1, 1991. This report is intended to acquaint our students, staff, industrial affiliates, and other colleagues and associates with the goals of the project. Section B contains the executive summary of the project, sections C and D describe the proposed equipment in greater detail, section F contains brief descriptions of a broad range of research projects proposed for the facility, and section H reviews the major results obtained from our previous institutional infrastructure grant, known as the TOPAZ project. Minor editing of sections B and F has been done to remove minor errors and omissions in the original proposal.

The material in this report is snapshot of our plans as of September 1990. While the general thrust of our proposed research is likely to evolve slowly, the reader should keep in mind that the details can change quickly.

PRISM stands for powerful research infrastructure for SIMD and MIMD experimentation.

B. EXECUTIVE SUMMARY

B.1. Introduction

This is a proposal to upgrade our department's infrastructure for experimental research in parallel computing. The requested PRISM facilities will allow our research programs in artificial intelligence, computer architecture, databases, mathematical programming, computer system performance, and scientific computing to continue to address issues and obtain results that are relevant to future parallel systems. The PRISM facilities will also serve as a catalyst for new experimental research projects in the above areas, as well as in programming languages and theoretical computer science.

The particular strengths of this proposal include: 1) the 27 important research problems that we will be able to address with the proposed equipment, 2) the discounts and external matching grants we have obtained for the equipment, and 3) the University of Wisconsin's commitment to the project, which is nearly 50% of the requested NSF support. In the following sections we summarize the rationale for the requested equipment (B.2), the requested equipment itself (B.3), the proposed research and staff credentials (B.4), the budget and institutional commitment (B.5), and the expected impact of the Institutional Infrastructure award (B.6).

B.2. Rationale for the Requested Infrastructure

Our existing parallel computing infrastructure, acquired with NSF and institutional matching funds under our Topaz project, includes two machines for research in MIMD parallel computing: 1) a 20-processor Sequent S-81 system with 40 megabytes of main memory, and 2) a 32-node Intel iPSC/2 with 32 disk drives and 256 megabytes of main memory. Significant research results made possible by the infrastructure are summarized in Section H of the NSF proposal. The existing infrastructure has served our research needs well. However, the Intel iPSC/2 is nearing obsolescence, and the Sequent S-81 architecture is limited to small-scale parallel computing.

The goals of the PRISM project are to obtain: 1) a sufficient number of processors to experiment with algorithm scalability, 2) sufficient memory and processing power to study real (i.e., large) applications on real data sets, and 3) architectures that have credible paths to achieving teraFLOPS processing power before the end of the decade. At least two types of architectures satisfy the third goal. One is the versatile MIMD multicomputer architecture, such as the Intel iPSC-series machines. The other is the more specialized data-parallel (or SIMD) architecture, such as the Connection Machine or MasPar MP-1. For both of these architectures, achieving high performance is a challenging research goal in many application

domains. There are also separate questions about the general applicability of SIMD architectures and the ease of programming and debugging large scale MIMD computations. While acquiring either of the above architectures would allow us to address the important corresponding issues, acquiring both architectures will allow us to address the questions in the context of the relative performance of the other architecture. Also, experimentation with both architectures will allow us to assess the extent to which it is possible and/or beneficial to "merge" the two architectures by incorporating (restricted forms of) one style of computing into the other type of architecture. The recent availability of powerful SIMD machines at relatively low cost, together with the equipment discounts and matching funds that we have been able to obtain, will allow us to acquire both types of architectures in configurations that satisfy our first two goals within the $2M budget for the PRISM project. We thus propose to replace our iPSC/2 machine with a larger and significantly more powerful MIMD system, and to acquire a powerful SIMD system.

B.3. The Requested Infrastructure

The requested equipment consists of high-performance SIMD and MIMD machines. In addition, we plan to develop programming tools for the MIMD machine, since programming environments currently available on such machines are relatively primitive. These machines and tools together provide a powerful research infrastructure for SIMD and MIMD computing (PRISM), and are briefly described below. Sections C and D contain further details about the equipment; Sections C and F contain further information about the tools.

The MIMD machine we currently have in mind is under development by the Intel Touchstone project. This machine, which will be referred to as the "iPSC/X" machine, is one full product generation beyond the iPSC/860, and two generations beyond our iPSC/2. The proposed configuration has 64 nodes, 1 gigabyte of main memory, and peak performance of 4-8 gigaFLOPS. It thus provides*: 1) a fundamental increase in processing power over our iPSC/2, 2) support for efficient broadcast communication that will be used by our proposed SIMD compiler, 3) hardware measurement facilities to support our proposed performance measurement tool, and 4) a mechanism for prototyping architecture modifications and new parallel architectures at a fraction of the normal cost (see Section F.6.2).

---

*We are intentionally vague about the exact nature of the iPSC/X hardware and software facilities since we have a nondisclosure agreement with the vendor.

The SIMD machine adds a new dimension to our infrastructure by supporting experimental research in the highly parallel SIMD style of computing. We currently envision purchasing the new, cost-effective MasPar MP-1, with 16,384 processors and 256 megabytes of main memory. This machine delivers 1.3 gigaFLOPS, or 26 billion instructions per second, of peak computing power and comes with excellent vendor-supplied software development tools.

Four programming tools that will be developed for the iPSC/X are: 1) a compiler for a SIMD language, 2) a parallel program debugging tool, 3) a program performance measurement tool, and 4) a batch scheduling system. The development of each proposed tool has a research component, either for finding the best solutions to problems that must be solved prior to its initial implementation, or for investigating problems that could lead to significant enhancements. The SIMD compiler is of particular interest since it will provide a platform for comparisons of the SIMD and MIMD architectures for executing programs expressed in a SIMD language, as well as for experimenting with hybrid SIMD/MIMD models of computation.

B.4. Proposed Research and Staff Credentials

The proposed research, discussed in Section F, is divided into the following eight areas:

1. parallel programming tools,
2. programming languages and environments,
3. databases,
4. parallel optimization algorithms,
5. scientific computing,
6. computer architecture,
7. artificial intelligence, and
8. models of parallel computation.

The MasPar MP-1 will support algorithm development for problem domains where SIMD is the natural programming paradigm, including graph algorithms for program integration [F.2.1], computer vision [F.7.1 and F.7.2], and learning methods [F.7.3]. The MP-1 will also support research in the applicability of SIMD computing to problem domains where MIMD algorithms are more widely used, including scientific computing [F.5.1 and F.5.2] and optimization problems [F.4.5 and F.4.6]. The iPSC/X machine will support the development of algorithms in various application domains for which MIMD is the natural paradigm, including databases [F.3.1, F.3.2, F.3.3], optimization problems [F.4.1 - F.4.4], and computer architecture evaluation tools [F.6.1 and F.6.3]. The iPSC/X will also support the development of software tools [F.1], and a cost-effective technique for prototyping new architectures [F.5.2]. Both machines

will support the development and evaluation of the SIMD compiler [F.1.1], comparisons of the two styles of computations [F.5.1, F.5.2], hybrid computations [F.4.5, F.4.6, F.7.2], and the development of improved conceptual models of parallel computing [F.8.1, F.8.2].

The Department has a ten-year history of research in distributed and parallel computing. Two of the larger recent projects are the Gamma parallel database machine research and the Center for Parallel Optimization. Participating faculty for PRISM all have outstanding research records and are internationally recognized for their contributions in their respective areas of expertise. Senior participating faculty members include a John von Neumann Chair of Mathematics and Computer Sciences (Mangasarian) and two University of Wisconsin Romnes Fellows (DeWitt and Kim). Kim also has an NSF Presidential Young Investigator (PYI) award. Recognition of participating faculty at the Associate and Assistant Professor levels includes two David and Lucille Packard Awards (Ramakrishnan and Reps), seven NSF PYI Awards (Bach, Carey, Hill, Horwitz, Ramakrishnan, Reps, and Vernon), and three ACM Distinguished Dissertation Awards (Bach, Condon, and Reps). Furthermore, the primary participants in the programming tools research, Larus, Livny, and Miller have each developed state-of-the-art tools that are used by colleagues within the Department as well as at other research institutions. These include the Condor batch scheduling facility for networks of workstations (Livny, DeWitt), the DeNet simulation language (Livny), the IPS parallel program measurement system (Miller), the "ae" parallel program analyzer (Larus). Staff credentials are further documented in Section G of the NSF proposal.

Results of the research supported by the proposed equipment are likely to include:

(1) numerous papers advancing the state-of-the art in future parallel computing,

(2) new algorithms for databases, optimization, simulating particles moving in viscous fluids, domain decomposition, computer vision, and learning,

(3) better conceptual models for performance analysis of parallel systems and parallel algorithm design,

(4) a SIMD compiler, a parallel program debugger, a program performance measurement tool, and a batch scheduling system for the iPSC/X,

(5) a high performance database machine capable of optimizing and executing highly complex workloads,

(6) a simulation testbed for parallel architectures,

(7) a prototyping testbed for parallel architectures, and

(8) an interactive program integration tool for practical programs.

B.5. Budget, Equipment Discounts, and Institutional Commitment

The budget for the proposed infrastructure includes a request for $2M from NSF, an institutional commitment from the University of Wisconsin of approximately $0.95M in new funding, and a matching grant from the DARPA/Intel Touchstone program. The institutional commitment, nearly 50% of the requested NSF support, indicates the high regard with which the University views our faculty and supports our need to acquire high performance parallel systems.

The cost of the 64-node iPSC/X system includes a 20% discount to the University of Wisconsin and an additional 15% University Partners' discount. The DARPA matching grant will further reduce the cost. The cost for the MasPar MP-1 includes a 32% discount through the MasPar DEC Data Parallel Research Initiative.

B.6. Conclusions

Our previous CER grants have been a major factor in the development of the Department and its research programs. One direct effect has been an increase in our annual research funding to over $5 million. Another effect has been the successful recruitment of a number of top young faculty, as noted in section B.4. In addition, interactions with industry have grown to include approximately fifteen industrial affiliates, as well as joint projects with IBM, DEC, AT&T Bell Laboratories, Cray Research, and Sequent. The CER grants were also a key factor in our receipt of state funding for a new $12 million building, constructed in 1985. The new grant is essential to maintain our tradition of state-of-the-art experimental research, and will allow us to investigate many important and timely problems in parallel computing.

C. INFRASTRUCTURE DESCRIPTION

We propose to acquire two new parallel computers, along with corresponding software, to support the PRISM project. One will be an SIMD computer with sufficient computational power and memory capacity to support experimentation with large-scale parallel applications. The other will be a scalable MIMD multicomputer with significantly more computational power than our present iPSC/2 system. In addition, we plan to develop a set of state-of-the-art software tools for the MIMD machine that will significantly enhance its utility for our research on high-performance parallel computing.

In this section we provide a brief overview of the proposed SIMD and MIMD equipment, followed by a discussion of the software infrastructure. We also summarize our personnel plans in support of the infrastructure. A more detailed description of our proposed equipment and software purchases, as well as budget details, are provided in Section D of the proposal. Further descriptions of the planned software tools can be found in Section F.

C.1. SIMD Equipment

The SIMD machine requested in this proposal is a data parallel computer, as typified by the Connection Machine or the MasPar MP-1. The SIMD machine is required for our proposed experimental research in data parallel algorithms for large-scale scientific computing, optimization problems, analysis of program dependence graphs, and computer vision. To support the PRISM effort, the proposed system must be configured with 16,384 processing elements (PEs), 256 megabytes of main memory, and a disk subsystem for use in solving large data-intensive problems. Parallel versions of C and Fortran are needed, as are software tools that support monitoring and debugging of parallel C and Fortran programs. In addition, the machine must be accessible through our local area network. Finally, since much of the proposed research on SIMD computing is focused on applications, we wish to avoid a machine for which tool development would be necessary before the bulk of the proposed research can begin. The machine identified in Section D as meeting these requirements with the best price/performance characteristics is a 16,384 PE MasPar MP-1 system with 256 megabytes of memory and a 2.7 gigabyte disk array. We propose to acquire this machine and its associated software in the first year of the grant at a total cost of $699,195, including $559,195 from NSF and $140,000 in matching funds from the University.

C.2. MIMD Equipment

The proposed MIMD machine is a message-passing multicomputer, similar to our iPSC/2 but significantly more powerful. The iPSC/2 has served our needs well, but its processors and

interconnection network are rapidly falling behind the state of the art. To support the MIMD portions of our research proposal, it is critical that we acquire a replacement for the iPSC/2 multicomputer. Our replacement hardware requirements include processing nodes that are 1-2 orders of magnitude faster than the iPSC/2's 80386 nodes, 4-8 times as much main memory, and 2-4 times the total number of processing nodes. In addition, if possible, we would like the machine to be open to the addition of new hardware; one of the architecture research projects proposed in Section F involves using the interconnection network of the machine to try out new design ideas for parallel architectures. Finally, the proposed database research for PRISM necessitates a well-configured I/O subsystem.

As explained in Section D, we propose to purchase a next-generation iPSC-series machine that we are referring to as the Intel iPSC/X machine. The iPSC/X machine will be based on a successor to the i860 processor, and it will have a mesh interconnection network. Based on information that we have received from Intel about this follow-on product (under a non-disclosure agreement), we are confident that our requirements and Intel's product offerings will coincide by the second year of the PRISM grant, which is when we propose to acquire the iPSC/X. We also understand that Intel is porting the Mach operating system to the iPSC product line as part of their Touchstone contract with DARPA, so we plan to use Mach as the operating system for the machine. Since prices are not available for the iPSC/X at this time, our budget is based on an iPSC/860 machine that approximates our requirements. We propose to acquire the MIMD hardware and software in the second year of the grant at a total cost of $1,548,752, including $750,482 from NSF and matching funds of $238,270 from DARPA and $560,000 from the University. The cost of this purchase will be spread out over Years 2 and 3 of the budget.

C.3. Infrastructure Software

Our system software requirements for the SIMD computer appear to be satisfied by the tools available from MasPar. We have similar software requirements for the MIMD multicomputer, but vendor-supplied multicomputer software is quite primitive. While C and Fortran are both available from vendors, substantial tools for parallel program debugging and performance tuning are not. Thus, our PRISM plans include the development of several new programming tools for use on message-passing multicomputers. These tools, described in Section F, include a SIMD language (C*) compiler for the MIMD multicomputer, a tool to aid in the debugging of message-based parallel programs, and a tool to assist in the performance evaluation and tuning of such programs. In addition, we plan to develop a batch scheduling system to make the multicomputer effective as a "parallel cycle server" for long-running parallel computations.

The SIMD compiler and the parallel program debugging tool will be useful prototypes available to users of the facility. The batch system scheduler and the parallel program measurement tool are envisioned as being closer to production-quality tools that will be in daily use on the system. Laboratory staff will be assigned to the tool development efforts, as discussed below.

C.4. Personnel

In addition to equipment, we are also requesting a small amount of NSF funding for personnel to support management of the project and development of the infrastructure software.

D. RESOURCE ALLOCATION

In this section we provide an overview of our current departmental research computing facilities followed by a detailed description and justification of the proposed SIMD and MIMD hardware and software acquisitions for the PRISM project. We also discuss the maintenance, user access, and space implications of the proposed facilities.

D.1. Existing Research Facilities

The Computer Sciences Department operates the Computer Systems Laboratory, which supports departmental computing needs for both research and instruction. The current research computing equipment can be broken down into three categories: workstations, file servers and general-purpose computers, and machines supporting our parallel computing research. The Department currently has about 225 Unix workstations allocated for research computing, including 48 DEC VAXstation 3200's, 121 DECstation 3100's, 2 DECstation 5000's, 22 IBM RT/PC's, 16 Sun 4/110's, 2 Sun 3/50's, 11 HP 360's, and 5 HP 345's. Five of the DEC workstations act as file servers; other file servers and general purpose machines include an 8-processor Sequent Balance, a DEC 5400, an HP 835, and 6 VAX 750's. In addition, we have numerous terminals and printers. All of our computers are connected via an Ethernet-based local area network, which also connects to the UW-Madison campus broadband network, to the Internet via CICNET/NSFNET, and to BITNET. In addition, the AI and VLSI Laboratories contain various special-purpose hardware and software for computer vision and computer architecture research.

Currently, our equipment for parallel computing research consists of two MIMD computer systems: a shared-memory multiprocessor and a message-passing multicomputer. The shared-memory multiprocessor is a 20-processor Sequent Symmetry (S-81) system with 2 megabytes of memory per processor and 1 gigabyte of disk storage. The multicomputer is a 32-processor Intel iPSC/2 system with 8 megabytes of memory and one 330-megabyte disk drive per processor; sixteen of the processors also have vector floating point boards. Both machines were acquired as part of the Department's NSF-sponsored Topaz project. Each runs vendor-supplied operating systems and compilers.

D.2. SIMD Computing Facilities

We have had detailed discussions with two companies, Thinking Machines and MasPar, regarding their current products, prices, and discounts. Both offer SIMD machines that fit the requirements outlined in Section C quite well. As of this writing, the MasPar MP-1 machine

Model Number	Description
Hardware :	
MP-1216A	MasPar MP-1 Data Parallel Computer*
DA-3014A	30 Series Parallel Disk Array**
IO-0002A	MPIOC 32MB Expansion Memory
WS-D002A	16 megabytes additional memory for 3520 workstation
WS-D006A	665 megabyte hard disk for 3520 workstation
WS-D004A	16 plane color graphics for 3520 workstation
Software:	
SW-2016A	MasPar Fortran 4-user software license
SW-2304A	MasPar Fortran license upgrade to 16 concurrent users
SU-0001A	MP-1 MPPE license upgrade to 16 concurrent users
SW-0002A	MasPar MPL license upgrade to 16 concurrent users

*Includes the MP-1200 Series Data Parallel Unit with 16,384 processing elements (PEs), 256 megabytes of PE memory, and an Array Control Unit with 1 megabyte of instruction memory. Also includes a Unix Subsystem based on a color VAXstation 3520 with 16 megabytes of memory, a 332 megabyte disk, a 296 megabyte streaming tape drive, and an Ethernet interface. Includes 4-user licenses for Ultrix and MPPE system software, and C and MPL compilers, with support for graphics, debugging, and program animation.

**Includes 4 data disks, 2.7 gigabytes of storage, and an MPIOC I/O controller with 8 megabytes of memory.

Table 1: SIMD Portion of Equipment Budget

appears to have significantly better price/performance characteristics than the Thinking Machines CM-2 offering, and both offer a comparable set of software tools, so the proposed budget is based on prices for an MP-1 machine from MasPar. We will, of course, revisit this decision when the time comes to actually order the SIMD machine. The SIMD portion of the total proposed hardware and software budget, based on figures from a recent MasPar price quotation, is summarized in Table 1. The MasPar prices include a discount available through the joint MasPar/DEC Data Parallel Research Initiative, and prices for the VAXstation 3520 workstation components are based on a DEC discount provided through this Initiative.

As indicated in Table 1, the initial MP-1 price includes all of the hardware and software for a well-configured 16,384 PE system. Included is a Unix Subsystem which serves to connect the MP-1 to a local network and hosts the MP-1 programming tools. The initial price also includes the MasPar system software, including compilers for C and the MasPar Parallel Language (MPL), as well as the MasPar Parallel Programming Environment (MPPE), which provides debugging and animation facilities for parallel programs. The hardware budget also includes a small disk array, 32 additional megabytes of I/O buffer memory for the disk array (recommended for performance reasons by MasPar), and expansion hardware for the Unix Subsystem (to increase its memory, disk, and graphics capabilities). The software budget includes the MasPar Fortran compiler as well as software license upgrades to permit up to 16 concurrent users of all system software components.

Model Number	Description
Hardware:	
iPSC/860 Model 64	iPSC/860 Parallel Supercomputer*
PSC860MMFRX8	64 8-megabyte i860 memory expansion modules
PSCMS42(x4)	8 additional I/O nodes with 16 disks (10.4 gigabytes total)
Software:	
PSCSRC	NX/2 and Concurrent File System source code
PSCNFS	Network File System (NFS) for the SRM

*Includes 64 i860 processing nodes with 8 megabytes of memory each. Also includes four 4-megabyte 80386 I/O nodes with 8 disks and 4.55 gigabytes, one 8mm tape drive, and a System Resource Manager (SRM) with 8 megabytes of memory, a 60 megabyte tape drive, a 1.2 megabyte floppy disk drive, and an Ethernet interface. Includes the iPSC/860 system software.

Table 2: MIMD Portion of Equipment Budget

D.3. MIMD Computing Facilities

We have had detailed discussions with Intel about their Touchstone series of machines, which they are developing as part of DARPA's Teraop project, and about their corresponding product plans. The current Intel product is the iPSC/860 hypercube, an i860-based machine with the same interconnection network as our present iPSC/2 machine. As described in Section C, we are proposing to acquire the iPSC/X machine, a next-generation machine that will replace the iPSC/860. However, since iPSC/X prices are not available at this time, our budget is based on an iPSC/860 machine that is configured to approximate the requirements stated in Section C. Compared to our current iPSC/2 machine, the proposed iPSC/860 configuration has twice as many processors, four times as much main memory, and processors that are an order of magnitude faster. It is our expectation that a similarly sized iPSC/X machine, with faster processors and even more main memory, will be available for a comparable price in 1992. For I/O, the budgeted iPSC/860 configuration has 24 disks attached through 12 I/O nodes.[†]

Table 2 shows the MIMD portion of the proposed budget. The figures are based on a recent Intel price quotation; they include a special discount that we have arranged with Intel. As indicated in the table, we have budgeted an iPSC/860 configuration that includes 64 i860-based processing nodes with 16 megabytes of memory each. As with the MasPar, the Intel system also includes a machine (the SRM) to run the iPSC/860 compilers and to host the

[†]We would prefer to take the 32 disks from our present iPSC/2 machine and install them directly on 32 processor nodes of the iPSC/X. This is not possible in the iPSC/860, but should be possible with the iPSC/X. Intel's I/O product plans for the iPSC/X are unclear at this time, though, so the current budget is based on existing i860 I/O options. We also hope to acquire more iPSC/X disks via future database research grants.

Budget Items & Funding Sources	1991-92	1992-93	1993-94	1994-95	1995-96	5-Year Cost
MasPar MP-1 Hardware & Software						
NSF, UW	$700K	—	—	—	—	$700K
Intel iPSC/860 Hardware & Software						
NSF, UW, DARPA	—	$800K	$750K	—	—	$1.5M
Total Hardware & Software						
NSF, UW, DARPA	$700K	$800K	$750K	—	—	$2.25M

Table 3: Annual Equipment-Related Expenditures

iPSC/860 on our Ethernet-based local area network. Turning to software, Intel will provide source code for the NX/2 operating system kernel and the Concurrent File System at no cost through the Intel University Partners Program. The only non-zero software budget item is an NFS license for the SRM to simplify file access.

D.4. Annual Equipment Expenditures

Table 3 shows how the proposed equipment-related expenditures are to be spread out over the life of the grant. We plan to purchase the SIMD machine in the first year of the grant. The MIMD purchase will be deferred until the second year, when the new Intel product is due to be available. We plan to spread the cost of the MIMD purchase over two years by making appropriate arrangements with Intel; they informally mentioned to us that such arrangements are possible. As indicated in the table, the University is providing matching funds for our equipment purchases. In addition, DARPA has agreed to provide a grant from their Touchstone effort for the iPSC/X machine. Finally, the maintenance figures in Table 3 are based on an estimate of 5% of equipment cost for annual MasPar maintenance and $10,000 per year for Intel maintenance, with both systems being covered by 1-year warranties during their first year of operation. The MasPar maintenance figure is based on past experience with hardware and software maintenance for the Crystal and Topaz CER grants. The Intel figure is based on a quotation from Intel for hardware self-maintenance (i.e., board swap maintenance) and an informal estimate for software upgrades.

In addition to the plans outlined here, we are also considering the possibility of linking the iPSC/X and MasPar machines using a HIPPI network. Both machines have HIPPI interfaces available, and a pair of these interfaces on the two machines would allow full-duplex communications at 800 megabits per second. Such a high-bandwidth connection between the machines would create an experimental SIMD/MIMD computing facility that could be used to conduct research on hybrid algorithms for applications where a hybrid computational style is advantageous. Based on a quotation from MasPar and a discussion with Intel, it appears that the machines could be connected in this way for a total of approximately $60,000. Thus, we plan

to seek additional funding, either from the University or via a separate grant, to obtain HIPPI interfaces for this purpose.

D.5. Access and Space Issues

The proposed SIMD and MIMD parallel computer systems will be accessed by researchers through our local area network. Both the MasPar MP-1 and Intel iPSC/X have workstations that serve as their front-ends. Each hosts compilers and other programming tools and interacts with user workstations via an Ethernet connection. We plan to run NFS on these front-ends in order to allow program and data files to be created and edited on user workstations and then accessed transparently from the front-ends.

No space renovation will be needed to accommodate either the SIMD or MIMD equipment that we are requesting.

E. BUDGET

(omitted)

F. RESEARCH

This section presents our proposed research in the following eight areas:

1. parallel programming tools,
2. programming languages and environments,
3. databases,
4. parallel optimization algorithms,
5. scientific computing,
6. computer architecture,
7. artificial intelligence, and
8. models of parallel computation.

In all, 27 projects are summarized in these 30 pages. In keeping with the infrastructure nature of this proposal, we have opted for brief presentations of many of our proposed projects, rather than focusing on a few of them in more detail. Within the limited space available for each project, we attempt to be relatively complete about describing the significance of the research problem and approach, as well as relatively terse but concrete about how the proposed research requires or is significantly enhanced by the proposed machines.

F.1. PARALLEL PROGRAMMING TOOLS

The tools research will be carried out by J. Larus, M. Livny, J. Naughton, B. Miller, and M. Vernon. Laboratory staff will be assigned to tool implementation as needed.

F.1.1. Compiling SIMD Programs for a MIMD Computer (J. Larus)

The two major parallel programming models embodied in commercial computers are SIMD (data parallelism) and MIMD (control parallelism). Languages that use the SIMD paradigm are typically closely tied to SIMD hardware that executes instructions in lockstep. This relationship may not be essential. If SIMD programs can be efficiently compiled for MIMD multicomputers, several benefits will accrue to programmers. First, SIMD languages offer many desirable properties for multicomputer programmability, including the shared memory and data parallel concepts, and computation reproducibility (i.e., ease of debugging). Second, a cross-compiler would make SIMD programs portable across a much wider range of computers and consequently available to a larger community of users. Finally, applications that contain efficient data-parallel portions as well as portions that require MIMD execution will be able to make effective use of both models. We thus propose to explore the question of whether programs expressed in a SIMD language can be compiled for efficient execution on a MIMD machine by writing a compiler for C* for the iPSC/X, and evaluating the compiler's performance against the comparably powerful MasPar MP-1.

The major previous work in this area was done by Quinn and Hatcher, who investigated compiling very simple C* programs for hypercubes and shared-memory multiprocessors [Quin88, Quin90]. Their preliminary results showed that small C* programs ran only slightly slower than hand-coded programs for the same algorithm. Although these results are promising, extensions to the compiler technology and language design are needed to determine whether the approach will work for real applications and for non-numeric application domains. For example, Quinn and Hatcher modified C* to require the programmer to specify the data partitioning. Techniques to automatically produce efficient data partitionings would significantly increase the value of the compiler.

Our initial approach will be to construct a C* to C translator that permits us to rapidly prototype and evaluate compiler algorithms. The compiler must produce code that executes asynchronously on the multicomputer, but appears to behave as if all executed in lockstep. On a distributed-memory machine, this difference is apparent only when processes communicate. We will investigate techniques to automatically restructure a program so that processors execute independently for relatively long intervals, and multiple messages are combined whenever possible. Chen's data partitioning techniques and other techniques for compiling sequential programs on distributed-memory computers might profitably be employed to achieve these goals [Chen86, Call88, Zima88].

The large-scale application programs for scientific computing, optimization problems, program integration, and computer vision to be developed by other participating faculty will provide excellent test cases for the prototype compiler. We will study these and other C* programs to understand how the language is used in practice and which features require special support. When the compiling techniques are sufficiently well-developed, we will modify the translator to generate native code for the computer and compare its performance against message-passing programs on the iPSC/X and C* programs on the MasPar.

The goal is to build a prototype C* compiler and runtime system that is practically useful. Additional benefits of this research include the following. First, the techniques developed for C* will be applicable to compiling other data parallel languages for MIMD computers. Second, these techniques can also be used to compile the data parallel subsets of non-SIMD languages, such as Fortran 90 and APL.

[Call88] Callahan, D. and K. Kennedy, "Compiling Programs for Distributed-Memory Multiprocessors," *Journal of Supercomputing*, **2**, 2, pp. 151-169 (October 1988).

[Chen86] Chen, M. C., "A Parallel Language and its Compilation to Multiprocessor" Machines," *POPL 13*, January 1986.

[Quin88] Quinn, M. J. and P. J. Hatcher and K. C. Jourdenais, "Compiling C* Programs for a Hypercube Multicomputer," *Proceedings of the ACM/SIGPLAN PPEALS 1988*, pp. 57-65, July 1988.

[Quin90] Quinn, M. J. and P. J. Hatcher, "Data Parallel Programming on Multicomputers," *IEEE Computer*, To Appear Sept. 1990.

[Zima88] Zima, H. P. and H. J. Bast and M. Gerndt, "Superb: A Tool for Semi-Automatic SIMD/MIMD Parallelization," *Parallel Computing*, **6**, pp. 1-18, 1988.

F.1.2. Parallel Program Debugging on Message-Based Multicomputers (B. P. Miller)

Flowback analysis is a powerful technique for debugging programs. It allows the programmer to examine dynamic dependencies in a program's execution history without having to re-execute the program. The goal is to present to the programmer a graphical view of the dynamic program dependencies. Our current system, called PPD [MCHb88, CHMI90, CMIN88], performs flowback analysis while keeping the execution time overhead low. We have extended the semantics of flowback analysis to shared-memory, parallel programs. Execution time overhead is kept low by recording only a small amount of trace during a program's execution. We use semantic analysis and a technique called *incremental tracing* to keep the time and space overhead low. Parallel programs have been accommodated by allowing the flowback dependences to span process boundaries; i.e., the most recent modification to a variable might be traced to a different process than the one that contains the current reference.

We propose to extend PPD to handle message-passing parallel programs with large numbers of processes. Our goal is to provide a prototype tool that can be used to develop applications on the iPSC/X. Flowback analysis tracks the flow of data by examining references and assignments to variables. In a message-passing system, data can flow through the additional paths created by the message channels. We will extend flowback analysis to track data in and out of message buffers, and to track the dynamic binding of sender to receiver. Flowback analysis has the advantage that the programmer need only look at a small part of the program at one time; this is ideal for applications that contain many processes. We will also examine issues in distributed control. We will apply our work in the area of breakpoint detection for distributed programs [MCHa88]. The iPSC/X will provide a facility to experiment with large-scale parallel programs on a message-passing architecture.

[CHMI90] J.-D. Choi and B.P. Miller, "Code Generation and Separate Compilation in a Parallel Program Debugger", in **Languages and Compilers for Parallel Computing**, Research Monographs in Parallel and Distributed Computing, MIT Press and Pitman Publishing, D. Gelernter, A. Nicolau, and D. Padua, eds., 1990.

[CMIN88] J.-D. Choi, B.P. Miller, and R. Netzer, "Techniques for Debugging Parallel Programs with Flowback Analysis", CS Tech Report #786, UW–Madison, August 1988. Submitted for publication.

[MCHa88] B.P. Miller, J.-D. Choi, "Breakpoints and Halting in Distributed Programs", Proc. of the 8th Int'l Conference on Distributed Computing Systems, San Jose, June, 1988.

[MCHb88] B.P. Miller, J.-D. Choi, "A Mechanism for Efficient Debugging of Parallel Programs", *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, Atlanta (June 1988).

F.1.3. Parallel Program Measurement on Message-Based Multicomputers (B. P. Miller)

IPS-2 is a performance measurement system for parallel and distributed programs [MILL90,YAMI90]. It allows application programmers to measure their parallel programs (both shared-memory and message-passing) and produce execution profiles, graphs, and high-level analyses. IPS-2 has a powerful graphical user interface. IPS-2 is in current use on VAX, DECstation, and Sequent Symmetry computers, and has been used for a wide variety of performance tasks. A project is underway (jointly with the San Diego Supercomputer Center) to port IPS-2 to the Cray Y/MP.

We plan to port the IPS-2 system to the iPSC/X computer. We anticipate using IPS-2 in production support for application programmers on this machine. IPS-2 has previously run in UNIX-based environments, so the effort required in porting IPS-2 will depend on the operating system provided on the iPSC/X. We will address issues such as the operating system interface and trace data collection. The iPSC/X designers at Intel have described new hardware support for program instrumentation that will simplify and streamline the implementation.

IPS-2 supports analyses, such as *Critical Path Analysis*, that help guide the programmer to the specific parts of the program that are slowing the execution. This guidance is particularly important in large-scale parallel programs. We will also explore new program visualization techniques to support very large numbers of processes.

[MILL90] B.P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, and S.-S. Lim, "IPS-2: The Second Generation of a Parallel Program Measurement System", *IEEE Trans. on Par. and Distr. Computing* **1**, 2, pp. 206-217 (April 1990).

[NEMI90] R. Netzer and B.P. Miller, "On the Complexity of Event Orderings for Shared-Memory Parallel Program Executions", *Proc. of the 1990 Int'l Conf. on Parallel Processing*, St Charles, Ill., August 1990.

[YAMI90] C. Yang, B.P. Miller, "Performance Measurement of Parallel and Distributed Programs: A Structured and Automatic Approach", *IEEE Transactions on Software Engineering* **15**, 12, pp. 1615-1629 (December 1989).

F.1.4. A Batch System for the iPSC/X (M. Livny, J. Naughton, and M. K. Vernon)

As a result of the concerted research effort that has been devoted to the area of parallel computation, a wide range of compute-intensive applications are beginning to exploit the parallelism provided by multicomputers. In order to place a multicomputer on the production floor, it has to be equipped with a batch system that will 1) manage its resources efficiently and fairly and 2) provide the users of these applications with easy access to its processing capacity. We propose to develop and prototype a batch system for the iPSC/X.

Since 1984 we have been engaged in an effort to develop a batch system for a cluster of workstations. The Condor distributed batch system [Litz88, Litz90], which is the result of this effort, provides researchers in our department with access to more than 200 workstations. The system employs a novel scheduling algorithm [Mutk87] to allocate computing capacity to over

40 users. With the help of a checkpointing mechanism, Condor effectively utilizes the processing capacity of the workstations and guarantees the completion of batch jobs.

The checkpointing mechanism of Condor can only handle jobs that consist of a single Unix process. Existing solutions to the distributed checkpoint problem, such as distributed database techniques and other checkpointing schemes that are based on synchronous logging of messages, are inadequate for use in a batch facility. Recently, we have developed and implemented a checkpointing scheme for shared-memory multiprocessors [Li90]. Currently, using the technique of special "marker" messages, we are extending our scheme to apply to message-based application checkpointing. We plan to implement the new scheme on the iPSC/X and use the proposed batch system to test it on real-life applications.

Is a recent study [Leut90] we showed that, in the absence of a-priori knowledge of processing demands, two types of multiprocessor scheduling policies have the best performance: 1) policies that give each job *an equal fraction* of the processing power of the multiprocessor, and 2) foreground-background policies that give each job an initial allocation of processing power, and then place it in a "background" queue. These results, along with our experience from the Condor system, will guide us in the design of the scheduling policy for the proposed batch system.

[Leut90] Leutenegger, S. T., and M. K. Vernon, "Performance of Multiprocessor Scheduling Algorithms", *ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, Boulder, CO, (May 1990).

[Li90] Li, K., Naughton, J., and J. Plank, "A real-time, concurrent checkpoint and recovery algorithm for parallel programs", *Proceedings of the 1990 ACM Symp. on Principles and Practice of Parallel Programming*, Seattle, Washington, pp. 90 - 99 (March 1990).

[Litz88] Litzkow M., Livny, M., and M. W. Mutka, "Condor - A Hunter of Idle Workstations" *Proc. of the 8th International Conference on Distributed Computing Systems,* San Jose, California, (June 1988).

[Litz90] Litzkow, M., and Livny, M., "Experience With the Condor Distributed Batch System", *IEEE Workshop on Experimental Distributed Systems*, Hunstville, AL. (Oct. 1990)

[Mutk87] M. W. Mutka, and M. Livny, "Scheduling Remote Processing Capacity In A Workstation-Processor Bank Network", *Proc. of the 7th Int'l. Conf. on Distributed Computing Systems,* Berlin, West Germany, September 1987.

F.2. RESEARCH IN PROGRAMMING LANGUAGES AND ENVIRONMENTS

Section F.2.1 proposes to develop data-parallel algorithms on the MasPar MP-1 for fast manipulation of program dependence graphs. Section F.2.2 proposes research in parallel execution of symbolic programs, including an investigation of language features that facilitate the detection of parallelism.

F.2.1. Data-Parallel Graph Algorithms for Program Integration (T. Reps and S. Horwitz)

By *program integration*, we mean the merging process that one has to go through when the source code of a program diverges into multiple variants (supporting different features, different operating systems, or incorporating different bug-fixes, for example). To date, the only available tools for assisting with program integration implement an operation for merging files as strings of text. When applied to programs, these tools are of limited utility—and potentially quite dangerous—because one has no guarantees about how the program produced behaves compared to the programs that were the inputs to the integration tool.

In contrast, our research has provided the foundation for creating a *semantics-based* tool for program integration [Horw89]. Semantics-based integration is based on the assumption that a difference in the *behavior* of one of the variant programs from that of the base program, rather than a difference in the *text*, is significant and must be preserved in the merged program. Although it is undecidable to determine whether a program modification actually leads to such a difference, it is possible to determine a safe approximation by comparing each of the variants with the base program. To determine this information, the integration algorithm described in [Horw89] employs a program representation that is similar to the *program dependence graphs* that have been used previously in vectorizing and parallelizing compilers [Kuck81, Ferr87]. The algorithm also makes use of Weiser's notion of a *program slice* to find just those statements of a program that determine the values of potentially affected variables [Weis84].

The steps of the integration algorithm (and other algorithms we have developed, such as for interprocedural slicing [Horw90]) involve computations on the dependence graphs that are used to represent programs. An example of such a computation is to compare several graphs to find vertices with differences in their adjacency lists of incoming edges. Our research will focus on fast data-parallel algorithms for such problems. Some aspects of our work seem particularly well suited for the SIMD model; in one of our program-integration algorithms (*e.g.*, see [Yang90]) the dependence graphs used are not only sparse, but of bounded in-degree.

Our current implementation is *incremental*; as a program is modified by a user, an auxiliary dependence graph that represents the program is updated to reflect the user's changes. However, this approach is only possible for very restricted languages, such as the one supported by our current prototype. As we move beyond toy languages—which will require that the system perform interprocedural data-flow analysis to build the auxiliary dependence graph—we have to fall back on non-incremental (batch) techniques. It would be unfortunate if the system were to lose its essentially instantaneous response rate for user-level operations.

MasPar-executed data-parallel algorithms for creating and manipulating dependence graphs might allow us to provide the *illusion* of an incremental implementation even though computations are actually performed from scratch.

[Ferr87] Ferrante, J., K. Ottenstein, and J. Warren, "The program dependence graph and its use in optimization", *ACM Trans. on Prog. Lang. and Syst.* **9**, 3, pp. 319-349 (July 1987).

[Horw89] Horwitz, S., J. Prins, and T. Reps, "Integrating non-interfering programs", *ACM Trans. on Prog. Lang. and Syst.* **11**, 3, pp. 347-387 (July 1989).

[Horw90] Horwitz, S., T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs", *ACM Trans. on Prog. Lang. and Syst.* **12**, 1, pp. 26-60 (Jan. 1990).

[Kuck81] Kuck, D.J., R.H. Kuhn, B. Leasure, D.A. Padua, and M. Wolfe, "Dependence graphs and compiler optimizations", *Conf. Record of the Eighth ACM Symposium on Principles of Programming Languages*, Williamsburg, VA, pp. 207-218 (Jan. 1981).

[Weis84] Weiser, M., "Program slicing", *IEEE Trans. on Softw. Eng.* **SE-10**, 4, pp. 352-357 (July 1984).

[Yang90] Yang, W., "A new algorithm for semantics-based program integration", Ph.D. dissertation, Computer Sciences Department, University of Wisconsin, Madison, WI, (Aug. 1990).

F.2.2. Parallel Symbolic Programming Languages (J. Larus)

Symbolic programs—for example, CAD tools, AI programs, and parallel compilers—manipulate large amounts of irregularly structured data, which is frequently composed of non-numeric quantities. Programs of this type, although computationally expensive, rarely run on parallel computers because of the great difficulty of expressing them in languages in which management of parallelism is left to the programmer. Authors of numeric programs face similar difficulties, but they are aided by compilers capable of translating Fortran programs for diverse parallel computers. We propose to study problems in compiling symbolic programs for parallel computers and to investigate new language designs to facilitate expressing these programs.

In the initial stage of this research, we developed a Lisp compiler that produced code for a shared-memory multiprocessor [Laru89, Laru90a]. This compiler demonstrated that some symbolic programs could be effectively compiled for parallel computers, but it did not permit detailed understanding of symbolic programs or the limitations of the compilation techniques. Recently, I have been using a new technique to study parallel symbolic programs [Laru90b, Laru90c]. This technique simulates the parallel execution of a program by using a memory reference trace from the program's sequential execution. It allows fast and easy characterization and comparison of complex programs and provides detailed information about their structure and data dependences. The technique has demonstrated that symbolic programs behave very differently from numeric programs and that existing parallel-compiler optimizations cannot improve the performance of symbolic programs.

Currently, we are using this new understanding of symbolic programs to develop new compiler optimizations and parallel execution strategies. In addition, we are exploring the design of programming languages that facilitate expression of symbolic algorithms in a manner suitable for compiling to parallel computers. However, our algorithms generate sufficient numbers of parallel tasks to easily oversaturate our current equipment. The iPSC/X would thus significantly enhance our ability to efficiently execute these algorithms, and the MasPar will permit exploration of the entirely new area of compiling symbolic programs for SIMD processors.

[Laru89] Larus, J. R., "Restructuring Symbolic Programs for Concurrent Execution on" Multiprocessors," *Computer Sciences Division Technical Report UCB/CSD 89/502,* University of California–Berkeley (May 1989).

[Laru90a] "Compiling Lisp Programs for Parallel Execution," *Journal of Lisp and Symbolic Computing,* To Appear.

[Laru90b] "Estimating the Potential Parallelism in Programs," *Proceedings of the Third Workshop on Languages and Compilers for Parallel Computing,* Irvine, California, (August 1990). To Appear MIT Press.

[Laru90c] "Predicting the Effects of Optimization on Parallel Programs," *Computer Sciences Technical Report #953,* University of Wisconsin–Madison (August 1990).

F.3. RESEARCH IN DATABASES

The database group will conduct research on the implications of MIMD-style parallel computing for three database problems: multi-user scheduling of data accesses and database resource usage (Carey, DeWitt, Livny), optimization of complex queries (DeWitt, Ioannidis), and the processing of database queries expressed in logic-based languages (Ioannidis, Naughton, Ramakrishnan).

The goal of the research proposed in this section is to develop practical algorithms that can ultimately be applied in commercial parallel database systems. Where possible, simulation will be used for initial algorithm evaluation. However, implementation and experimentation are necessary to ensure the practicality and scalability of the most promising solutions. We plan to port Wisconsin's Gamma parallel database system to the iPSC/X under the Mach operating system in order to provide the required testbed for each of the proposed projects. The iPSC/X will be an ideal platform for this work, providing a state-of-the-art "shared nothing" hardware base that is comparable in both CPU performance and memory size to those of commercial parallel database systems.

F.3.1. Multiuser Scheduling in Parallel DB Machines (M. Carey, D. DeWitt, M. Livny)

Over the past seven years, a number of successful database machines have been built by combining highly parallel dataflow query processing algorithms with fairly conventional message-based multicomputers where one or more disks are typically attached to each processor [DeWi90a]. Notable research prototypes include the Wisconsin Gamma system

[DeWi90b], Bubba at MCC, and Arbre at IBM Almaden; significant commercial offerings are available from Teradata and Tandem. In each of these systems, parallelism is obtained by declustering [Livn87] the database relations over many nodes and then employing parallel algorithms to execute each of the basic relational database operators involved in a query. Pipelined execution of multiple operators of a complex query can provide additional parallelism.

Multiuser benchmarks of the Tandem, Teradata, and Gamma systems have shown that parallel database machines can be very effective for both transaction processing workloads, where inter-transaction parallelism is needed to handle many simple transactions, and for complex batch workloads, where successful application of intra-transaction parallelism is the key to good performance. However, processing a mix of small transactions and complex, on-line queries poses several challenging problems. One problem is that complex queries often require many locks, and hold them for long periods of time, impeding the progress of short transactions. The current commercial solution is "browse mode" execution, which allows complex queries to obtain approximate answers by reading inconsistent data. Clearly, this is not sufficient for all applications. For conventional database management systems, *multiversion* concurrency control algorithms are an alternative solution, and simulations indicate that they may indeed be effective [Care86b]. We intend to explore the use of multiversion locking algorithms in the context of a parallel database machine. Issues to be studied include how to distribute versions of relations across the processing nodes and how to integrate versions with Gamma's algorithms for data replication and recovery [DeWi90b, Hsia90].

An important related problem, also beyond the current state of the art, is how to schedule the many physical resources of a parallel database machine at runtime. In particular, decisions made by the query optimizer are based on cost estimates, which in turn are based on assumptions about the amount of available memory and the number of available processors. In fact, neither can truly be known until query execution time, long after the query optimizer has generated and compiled a plan for executing the query. But what if the system is incrementally presented with a workload containing many queries of varying sizes and complexity? Runtime scheduling issues include how much memory to allocate to the operators of each query and, for intermediate query results, over how many processors to decluster the results in order to parallelize the execution of subsequent query operators. Moreover, some queries in the workload may be more important than others, and should thus be given higher priority [Care89]. Lastly, replicated data provided by Gamma's approach to high availability [Hsia90] provides an opportunity to use dynamic load balancing (as in [Care86a]) to correct load imbalances caused by skewed or hot data. We plan to design and evaluate adaptive algorithms for making the runtime scheduling decisions needed to effectively cope with complex query mixes.

[Care86a] Carey, M., and H. Lu, "Load Balancing in a Locally Distributed Database System," *Proc. of the 1986 ACM SIGMOD Conf.*, Washington, D.C., pp. 108-119 (May 1986).

[Care86b] Carey, M., and W. Muhanna, "The Performance of Multiversion Concurrency Control Algorithms," *ACM Trans. on Computer Systems*, Vol. 4, No. 4, pp. 338-378 (November 1986).

[Care89] Carey, M., R. Jauhari, and M. Livny, "Priority in DBMS Resource Scheduling," *Proc. of the 15th VLDB Conf.*, Amsterdam, The Netherlands, pp. 397-410 (August 1989).

[DeWi90a] DeWitt, D., and J. Gray, "Parallel Database Systems: The Future of Database Processing or a Passing Fad?," *SIGMOD Record*, to appear (December 1990).

[DeWi90b] DeWitt, D., et al, "The Gamma Database Machine Project," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 1, pp. 44-62 (March 1990).

[Hsia90] Hsiao, H., and D. DeWitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines," *Proc. of the 6th IEEE Data Engineering Conf.*, Los Angeles, CA, pp. 466-475 (February 1990).

[Livn87] Livny, M., S. Khoshafian, and H. Boral, "Multi-Disk Management Algorithms," *Proc. of the 1987 ACM SIGMETRICS Conf.*, Banff, Alberta, Canada, pp. 69-77 (May 1987).

F.3.2. Parallel Query Optimization (D. DeWitt and Y. Ioannidis)

One of the primary reasons that relational database systems have been such a commercial success is the effectiveness of their query optimization module. The input to this module is a query q given by the user. The optimizer's purpose is to select the most efficient algorithm to access the relevant data and answer the query. Specifically, for the relational model, a *strategy* to answer a query q is a sequence of relational algebra operators (e.g., select, project, join, union) applied to the relations in the database that eventually produces the answer to q. The specification of a strategy also includes the method (algorithm) that will be used for each operator (e.g., hash-join, merge-scan, or nested-loops for a join operator) and the indices that will be used, if any, to access each relation (e.g., B^+-trees or hash tables on relation attributes). For every query, there are many choices for the sequence of operators, the methods that implement them, and the indices used, thus resulting in a large strategy space that the query optimizer needs to search for its least costly member(s).

Future applications pose several new challenges to database systems. Queries will need to access much larger data sets than they do today (increasing from megabytes to terabytes). Also, users will need to ask significantly more complex queries than what has traditionally been the case (moving from single, nonrecursive queries with few joins, to multiple, possibly recursive queries with tens or hundreds of joins). Parallelism is the only way for database systems to cope with such unprecedented levels of complexity. Current query optimization technology, however, is inadequate for a parallel environment, because the space of alternative strategies is, in many respects, very different from what is seen in conventional systems today. The two most important points of difference, which we plan to address in our work, are the specific strategies that are included in the search space and the size of the search space.

A significant amount of work exists on parallel query processing strategies, especially in the context of multiprocessor database machines [Schn90]. Incorporating these strategies in a query optimizer's search space is nontrivial. First, a parallel query processing strategy includes additional parameters, e.g., distribution of subqueries to processors and synchronization points for interleaving of parallel and sequential execution. Extensive studies are needed to determine which parameters should have their values chosen by the optimizer and which ones can be pre-specified. Moreover, due to the large number of such parameters, modeling the space of alternative strategies in a regular, algebraic way is difficult. Second, new cost models need to be developed and validated for alternative parallel strategies such as those proposed in [Schn90]. Third, the problem of cost estimation is even harder than in the sequential case. Not only are new estimation techniques needed, but error propagation becomes a more pressing problem due to the expected decreased accuracy of the estimates. Alternatives like interleaving optimization and execution need to be investigated.

The availability of a wide variety of parallel strategies, each with several parameters whose values are to be determined by the query optimizer, results in a significant increase in the size of the strategy space that needs to be explored. The anticipated increase in query complexity has a similar effect as well. Most conventional systems use sequential, heuristically pruning, semi-exhaustive algorithms to search the strategy space, but this will be inadequate in the future. Randomized optimization algorithms appear to be the most promising alternative, so we plan to study several such algorithms in a parallel environment. We have already experimented with sequential implementations of three such algorithms, *simulated annealing*, *iterative improvement*, and *two-phase optimization*, on search spaces of sequential strategies [Ioan87, Ioan90]. We plan to incorporate parallel strategies in these search spaces and also to adapt the optimization algorithms themselves for parallel execution on the iPSC/X. Some of these methods have very straightforward parallelizations, whereas others do not, making the parallelization task very challenging. We also plan to study *genetic algorithms* [Gold89], which have been very effective in other disciplines and lend themselves to relatively easy parallelizations. This research ties in with the work on genetic algorithms discussed in section F.4.3 below.

[Gold89] Goldberg, D., **Genetic Algorithms in Search, Optimization, and Machine Learning,** Addison-Wesley, Reading, MA, (1989).

[Ioan87] Ioannidis, Y., and E. Wong, "Query Optimization by Simulated Annealing," *Proc. of the 1987 ACM SIGMOD Conf.,* San Francisco, CA, pp. 9-22 (June 1987).

[Ioan90] Ioannidis, Y., and Y. Kang, "Randomized Algorithms for Optimizing Large Join Queries," *Proc. of the 1990 ACM SIGMOD Conf.,* Atlantic City, NJ pp. 312-321 (May 1990).

[Schn90] Schneider, D. and D. DeWitt, "Tradeoffs in Processing Multi-Way Join Queries via Hashing in Multiprocessor Database Machines," *Proc. of the 16th VLDB Conf.*, Brisbane, Australia, pp. 469-480 (August 1990).

F.3.3. Parallel Evaluation of Data Intensive Logic Programs (Y. Ioannidis, J. Naughton, and R. Ramakrishnan)

Logic provides a very high-level language that is amenable to extensive optimization since the meaning of a program is defined independently of its execution. For example, if t denotes the transitive closure of a binary relation e, then it may be defined thus:

t(X,Y) :- e(X,Z), t(Z,Y).
t(X,Y) :- e(X,Y).

The procedural interpretation of this example in languages such as Prolog is that the clauses (rules) with t on the left side provide a program defining the procedure t. A call to t is evaluated by invoking a rule and generating (sub) calls for literals on the right side. In contrast, the rules can also be read as statements in standard logic: "right side implies left side."

As with conventional database queries, one may optimize a logic program by finding a more efficient but (logically) equivalent version, and there is an extensive literature on this subject [Banc86]. The user is able to specify the program at an abstract level, and the system assumes the responsibility for efficient execution, including parallelizing of the program.

Evaluation methods in the literature can be understood in terms of the procedural semantics, which is recursive identification of subgoals. A method is deemed more parallel if it is able to identify and work on more goals earlier. We recently proposed an abstract model that seeks to capture this measure of parallelism [Rama90]. There are two main sources of parallelism in a logic program: different rules can be evaluated concurrently, and subgoals generated from different literals in a rule can be solved in parallel. (The latter, of course, depends on the order in which literals are examined, as solutions for one literal can be used to restrict the subgoals generated from subsequent literals.) With these two sources of parallelism, logic programs tend to exhibit a much higher level of explicit parallelism than that which exists in traditional, imperative programming languages.

Currently, virtually all parallel implementations are based upon top-down approaches, possibly due to the traditional belief that bottom-up approaches cannot focus the computation effectively. However, it is now known that program transformation can be used to achieve the same restriction for bottom-up evaluation [Rama88]. Further, for a given choice of literal order, the proposed approach is maximally parallel according to the measure mentioned above [Rama90]. This provides strong motivation for investigating parallel implementations based upon bottom-up techniques.

The above "maximally parallel" result is an idealized picture that ignores many implementation overheads. We plan to use the iPSC/X to investigate and compare implementation-related issues for both top-down and bottom-up approaches to parallelizing logic programs. Such issues include tradeoffs in data representation, scheduling, and (in the context of data intensive programs) accessing disk-resident data, all of which could have a significant impact. For example, many top-down approaches deliberately restrict the number of subgoals solved in parallel in order to minimize scheduling overheads. The scheduling overhead may be lower for bottom-up approaches since the goal-subgoal structure need not be explicitly maintained, and simple techniques for distributing the computation across multiple processors have been proposed (e.g., [Wolf88]). On the other hand, this has been achieved at the cost of introducing additional body literals into rules.†

[Banc86] Bancilhon F., and R. Ramakrishnan, "An Amateur's Introduction to Recursive Query Processing Strategies," *Proc. of the 1986 ACM SIGMOD Conf.*, Washington, D.C., pp. 16-52 (May 1986).

[Rama88] Ramakrishnan, R., "Magic Templates: A Spellbinding Approach to Logic Programs," *Proc. of the Int'l. Conf. on Logic Programming*, Seattle, pp. 140-159 (August 1988).

[Rama90] Ramakrishnan, R., "Parallelism in Logic Programs," *Proc. of the ACM POPL Symp. on Programming Languages*, San Francisco, pp. 246-260 (January 1990).

[Wolf88] Wolfson O., and A. Silberschatz, "Sharing the Load of Logic Program Evaluations," *Proc. of the 1988 ACM SIGMOD Conf.*, Chicago, IL, pp. 329-336 (June 1988).

F.4. RESEARCH IN PARALLEL OPTIMIZATION ALGORITHMS

Since the Dantzig-Wolfe decomposition method of 1960, mathematical programmers have been developing techniques to partition large-scale problems into smaller optimization problems that could be solved independently. However, the practicality of solving real-world linear and nonlinear programs with hundreds of thousands or millions of variables remained equivocal until the 1980's produced new developments in both algorithms and parallel architectures. Starting with our original CER grant (Crystal), a major focus of the research of our mathematical programming group has been the development of new parallel algorithms for a variety of difficult optimization problems, including linear programs, linear complementarity problems, network optimization, discrete optimization, and stochastic programs. While our current facilities have allowed us to test our algorithms on medium-sized problems and some specially-structured large problems, we require parallel systems with supercomputer performance and memory capacity to carry out our research for large-scale, real-world problems.

---

† Additional rules are also introduced, but their application corresponds to the top-down generation of goals, and can be viewed as implicit in top-down approaches as well.

F.4.1. Parallel Constraint Distribution (O. L. Mangasarian and M. C. Ferris)

We have proposed a novel parallel algorithm for handling constrained optimization problems with a very large number of constraints [DeLe90]. The idea is to break the constraints into blocks and distribute them among parallel processors while giving each processor a properly modified objective function. The key to an efficient implementation of the algorithm is the modification of the objective function that is given to each processor. Currently the most promising modification consists of an augmented Lagrangian objective [Ferr90]. For linear programs with a number of constraints far exceeding the number of variables, linear speedup is possible under suitable realistic assumptions. A system with the speed and local memory size of the proposed iPSC/X would allow us to solve hitherto intractable problems, as the only information passed among processors is Lagrangian multiplier information for the relatively small-sized constraint blocks assigned to each processor. Massively parallel architecture, such as that of the MasPar, on the other hand, have the potential of rapidly solving problems with an enormous number of constraints via the allocation of small blocks of constraints to each processor.

[DeLe90] De Leone, R. and O. L. Mangasarian "Parallel proximal point decomposition of linear programming constraints", *SIAM Annual Meeting*, Chicago, Illinois, July 16-20, 1990.

[Ferr90] Ferris, M. C. and O. L. Mangasarian "Parallel distribution of convex programming constraints", *Symposium on Parallel Optimization 2*, Madison, Wisconsin, July 23-25, 1990.

F.4.2. Parallel Pattern Separation via Linear Programming (O. L. Mangasarian)

Recently [Benn90] we have shown that a practical method for pattern separation that is based on linear programming, the multisurface method (MSM) [Mang68], is equivalent to training a neural network with some predetermined weights. MSM has been very successfully used for the diagnosis of breast cancer at University of Wisconsin Hospitals [Mang89]. During the past 17 months it correctly diagnosed 165 out of 166 cases. A number of approaches exist for parallelizing our underlying training algorithm to allow us to handle large-dimensional problems with very large data sets. Examples of such large problems are determination of solvency of banks from extensive financial records, making important oil drilling decisions from extensive seismic data, and making critical decisions based directly on medical imaging. At the simplest level is parallelization of the underlying linear programming training algorithm. This can be achieved in a number of ways. One way is to parallelize the linear programming algorithm itself (whether simplex-based or interior-point-based). Another parallelization can be achieved by the parallel solution of the essentially independent linear programs at each step of the MSM. The number of these linear programs increases linearly with problem dimension, and hence parallelization would guarantee linear speedup. At a higher level we have the possibility

of a parallel training algorithm operating on disjoint or partly overlapping data sets. The iPSC/X architecture is very suitable for these parallelizations and would allow the solution of extremely large pattern separation problems in considerably faster time.

[Benn90] Bennett, K. P. & O. L. Mangasarian, "Neural network training via linear programming", *Computer Sciences Tech Report 948*, University of Wisconsin-Madison (July 1990).

[Mang68] Mangasarian. O. L., "Multi-surface method of pattern separation", *IEEE Transactions on Information Theory*, **IT-14**, pp. 801-807 (1968).

[Mang89] Mangasarian, O. L., R. Setiono & W. H. Wolberg, "Pattern recognition via linear programming: Theory and application to medical diagnosis", *Computer Sciences Tech. Report 878*, (October 1989). To appear in *Proceedings of SIAM Workshop on Large-Scale Numerical Optimization*, Ithaca, New York.

F.4.3. Genetic Algorithms for Combinatorial Optimization Problems (M.C. Ferris)

There has been an increasing interest in the use of search techniques which contain a stochastic element as methods for solving hard combinatorial optimization problems arising in practical applications. We shall consider the use of genetic algorithms to solve these problems. One of the major differences between genetic algorithms and other approaches is that genetic algorithms operate using a whole population of individuals, and in this sense they have a kind of natural parallelism not found, for example, in simulated annealing or Tabu search techniques. The problems of communication and synchronization have not previously been considered in detail. In this context, to overcome synchronization penalties we will implement a parallel asynchronous version of the algorithm, and to reduce communication we will construct an algorithm based on a local neighborhood construct [And90]. Another problem we wish to address is how to incorporate constraints (such as precedence relations) into these methods. Several techniques for treatment of these constraints are available, each of which is computationally intensive. The iPSC/X has the processing power we need to effectively handle large population sizes.

[And90] Anderson, E.J. and Ferris, M.C. "A genetic algorithm for the assembly line balancing problem", *Proceedings of the Integer Programming/Combinatorial Optimization Conference*, Waterloo, Ontario, Canada, May 28-30, 1990.

F.4.4. Parallel Solution of Large-Scale Integer Linear Programs (R. De Leone)

Branch and bound algorithms are well-known and effective methods for discrete optimization problems. They are implicit enumeration methods that are conveniently represented by a finite rooted tree. Processing at each node of the tree involves completion of various tasks such as the computation of a bound on the optimal solution value, fixing some of the variables, branching, etc.

The collection and distribution of aggregate node information as well as the tasks for a particular node can be performed by different processors in parallel [Rush90]. At the same

time, other processors can improve the current upper and lower bounds by using ad hoc heuristics or algorithms based on Lagrangian relaxation or dynamic programming based methods. By efficiently combining the information from all the processors, the search space can be substantially reduced and therefore discrete problems with very large numbers of variables can be efficiently solved on the iPSC/X. Through iPSC/X implementations, we will investigate the effectiveness of such parallel approaches in solving specially structured integer programs. In particular, we will concentrate on large scale knapsack problems and we will test the algorithm on problems arising in digital system design [DeLe90]. We will also investigate parallel dynamic programming algorithms for knapsack problems. Since at each stage of the algorithm relatively simple tasks need to be performed, this method can be efficiently implemented on SIMD machines.

[Rush90] Rushmeier, R. and Nemhauser, G. "A Cooperating Multiple Search Approach for Parallel Integer Programming", *Symposium on Parallel Optimization 2*, Madison, Wisconsin, July, 1990.

[DeLe90] De Leone, R., Rajiv, J., Strauss, K. "Solution of Multiple-Choice Knapsack Problem Encountered in High-Level Synthesys of VLSI Circuits", *Computer Sciences Department Technical Report 980*, UW-Madison,(1990).

F.4.5. Hybrid Algorithms for Structured Optimization (R. De Leone and R.R. Meyer)

Most large-scale optimization problems have block structure (for example, blocks of constraints corresponding to multiple time-periods or commodities). This allows them to be decomposed iteratively via a variety of methods [Dant63, Sch90, DeLe90] into collections of approximating subproblems to which a coordination process is then applied.

A natural application of a hybrid parallel system, which supports both SIMD and MIMD modes, would allocate (completely or partially) the solution of the subproblems to the MasPar system where an algorithm such as parallel pricing (see the following section on hybrid algorithms for network flow) or SOR [DeLe90] or a network relaxation method [Bert89] would be employed. The coordination of the subproblem solutions at each iteration would be performed on the iPSC/X. The coordination process could also be overlapped with the solution of the subproblems (by using prior update information) or could use partial information from the current subproblems . MIMD-oriented methods of this type on both shared-memory [Sch90] and multicomputer systems [DeLe90] have proved to very effective in providing fast solutions of large real-world multicommodity problems.

[Bert89] Bertsekas, Dimitri P. and John N. Tsitsiklis, **Parallel and Distributed Computation** , Prentice-Hall, Englewood Cliffs, NJ, (1989).

[DeLe90] De Leone, R. "Solution of Very Scale Networks by Successive Overrelaxation ", *SIAM Annual Meeting*, Chicago, Illinois, July 16-20, 1990.

[Dant63] G. B. Dantzig, **Linear Programming and Extensions**, Princeton University Press, Princeton, NJ (1963)

[Sch90] Schultz , G. and R. Meyer, "A Structured Interior Point Method", *Computer Sciences Department Technical Report #934* , University of Wisconsin-Madison, (1990), submitted to proceedings of the Symposium on Parallel Optimization 2, Madison, Wi, 1990.

F.4.6. Hybrid Algorithms for Network Flow Problems (R. R. Meyer)

Parallel asynchronous extensions of the network simplex method have demonstrated the capability of solving very large network flow problems (e.g., hundreds of thousands of variables) in just a few minutes even on modest shared-memory systems [Clar90]. In order to achieve the solution of problems of millions of variables in real time, the facilities of a hybrid parallel computing system could be effectively utilized by performing the complex "pivot" operation of the network simplex method [Kenn80] in MIMD mode while using SIMD to do the "pricing" operation on large numbers of arcs (one or more arcs per SIMD processing element). Research is needed on selection strategies for the arcs to be priced and on the relative efficiency of traditional pricing versus approximation to maximum improvement strategies. For example, some data on basis tree structure (such as, portions of backpaths) could be used in SIMD mode, or a parallel evaluation of relevant backpaths for a small set of arcs that priced attractively in SIMD mode could be done on MIMD mode . It would also be of interest to try variants of this strategy for general linear programs.

[Clar90] Clark, R. and R. Meyer, "Parallel Arc-Allocation Algorithms for Optimizing Generalized Networks", *Annals of Operations Research* , **22,** pp. 129-160 (1990).

[Kenn80] Kennington, J. and R. Helgason, **Algorithms for Network Programming** , John Wiley, New York, (1980).

F.4.7. Parallel Decomposition of Large Stochastic Programs (S. M. Robinson)

In modeling resource allocation under uncertainty using discrete probability distributions, one encounters very large block-separable convex programming problems with coupling constraints. Current computational algorithms for this class of problem are so slow as to be impractical in many cases. For mathematical background on one such procedure (the partial inverse decomposition) see [RoWe87] and [Rob90], and for a detailed treatment of an application in energy capacity planning see [Dan89]. Since these problems are block-separable except for the coupling constraints, there is a conceptually simple decomposition algorithm based on dualizing with respect to the coupling constraints and maximizing the resulting nonsmooth dual objective using a bundle of dual subgradients. For a description and justification of this method, see [Rob89]. This general technique is independent of the actual method used to generate the bundle. For example, in [Dan89] the L-shaped method of Van Slyke and Wets [VSW69] is used, but methods of the bundle class such as the bundle-trust region algorithm [Zow89] appear to be much more efficient.

We plan to investigate the combination of fast bundle methods for decomposing the problems with simultaneous parallel solution of the decomposed subproblems using the iPSC/X. The experiments of Medhi [Med90] using the less efficient Lemaréchal bundle method and the Crystal multicomputer at Madison showed an efficiency of more than 60%, and thus with the BT algorithm and the iPSC/X we expect to achieve enough speedup to make previously intractable planning problems solvable.

[Dan89] Dantzig, G.B. et al., "Decomposition Techniques for Multi-Area Generation and Transmission Planning Under Uncertainty", *Report EL-6484*, Electric Power Research Institute, Palo Alto, CA (1989).

[Med90] Medhi, D., "Parallel Bundle-Based Decomposition for Large-Scale Structured Mathematical Programming Problems", *Annals of Operations Research* **22**, pp. 101-127 (1990).

[Rob89] Robinson, S.M. "Bundle-Based Decomposition: Conditions for Convergence", in **Analyse Non Linéaire**, H. Attouch, J.-P. Aubin, F. Clarke, and I. Ekeland, eds., Gauthier-Villars, Paris, pp. 435-447 (1989).

[Rob90] Robinson, S.M., "Extended Scenario Analysis", Manuscript, April 1990, forthcoming in *Annals of Operations Research.*

[RoWe87] Rockafellar, R.T. and R. J.-B. Wets, "Scenarios and Policy Aggregation in Optimization Under Uncertainty", Manuscript, 1987, forthcoming in *Mathematics of Operations Research.*

[VSW89] Van Slyke, R. and R. J.-B. Wets, "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming", *SIAM Journal on Applied Mathematics* **17**, pp. 638-663 (1969).

[Zow89] Zowe, J., "The BT-Algorithm for Minimizing and Nonsmooth Functional Subject to Linear Constraints", in **Nonsmooth Optimization and Related Topics**, F. Clarke, V. Dem'yanov, and F. Gianessi, eds., Plenum Press, New York (1989).

F.5. RESEARCH IN SCIENTIFIC COMPUTING

F.5.1. Complex Microstructures in a Viscous Fluid: MIMD versus SIMD (S. Kim)

Dynamics of particles (size from 0.01 to 100 microns) immersed in a liquid incorporate N-body physics and solution of the Navier-Stokes equations for the motion of the intervening liquid. Given the mobility or link between forces and particle motion(s), mathematical models for aggregation kinetics (suspension stability) to rheology (viscosity and other flow properties) follow. Applications include dynamics of globular proteins (enzyme kinetics and drug design) [McC87], processing of silver halide grains in photographic emulsions, and aggregation kinetics of bentonite in oil well drilling fluids.

At the microstructure level viscosity dominates and the nonlinear (inertial) terms in the Navier-Stokes equations become negligible. The resulting Stokes equations can be recast (after derivation of the Green's function and integral representation) as a two-dimensional integral equation on the particle surfaces. We have developed [Kar89, Kim90] a new integral representation in which the integral operator is a contraction mapping; iterative methods converge to the unique fixed point. The matrix obtained by discretization of the kernel of the (boundary) integral equation has element ij as a simple function of the inner and outer products of the 3-D vector joining boundary elements i and j. Indeed, rows and/or columns can be constructed by

data parallel operations [Kim90, Ch.19] if the boundary element geometry is stored in memory. Fortunately, the geometry of a single particle is usually described by a small number of elements which can be replicated as needed. 3-D simulations require thousands (and millions preferably) of particles, but this part of the algorithm replicates in a data parallel manner, making this a candidate for SIMD-style computing. Since our large-scale suspension simulation is amenable to both SIMD and MIMD style computing, comparisons will be explored: SIMD on the MasPar, simulated SIMD on the iPSC/X and MIMD on the iPSC/X.

Finally, interactions between distant particles resemble point-particle interactions, which in turn are further attenuated (screened) by the intervening suspension. The computational analog will be explored by zeroing out entire blocks, while blocks corresponding to interactions between nearer neighbors can be treated by multipole expansion, analogous to the Greengard-Roklin algorithm. The frequency and size of messages are thus greatly diminished so that large scale simulations will be feasible on message-passing architectures.

[McC87] McCammon, J. and S. Harvey, **Dynamics of Proteins and Nucleic Acids**, Cambridge University Press, Cambridge (1987).

[Kar89] Karrila, S., Y. Fuentes and S. Kim, "Parallel Computational Strategies for Hydrodynamic Interactions between Rigid Particles of Arbitrary Shape in a Viscous Fluid", *Journal of Rheology* 33, 8, pp. 913-947 (August 1989).

[Kim90] Kim, S. and S. Karrila, **Microhydrodynamics: Principles and Selected Applications**, Butterworths, London (1990).

F.5.2. Domain Decomposition (J.C. Strikwerda and S. V. Parter)

The method of domain decomposition can be used with a parallel computing environment in several ways, e.g., [Glow88]. For example, in an MIMD environment, different domains can be assigned to different processors. Also, the interpolation of data between grids could be assigned to different processors. In an SIMD environment, the regular arrangement of grid points on each sub-domain and the local nature of the finite difference schemes makes possible several SIMD approaches. The simplest approach is to use SIMD algorithms on each sub-domain in turn. One algorithm that may be used in this context is Multigrid, which uses a strategy of grid refinement and operators on a succession of grids each with fewer points. Multigrid may be efficiently implemented in several ways on the MasPar. For example, iterations on different domains could be done simultaneously so as to maximize utilization.

F.5.2.1. Incompressible Fluid Dynamics

This research is devoted to developing methods to efficiently and accurately compute complex incompressible fluid flows in non-simple domains. The primary method is to use domain decomposition to separate the domain into smaller sub-domains with simple shapes

such as rectangles and annuluses. On each sub-domain the equations are approximated by finite difference schemes, and these are solved by iterative methods. The values of the velocity are communicated from one domain to another by interpolating values to the boundary.

Strikwerda has developed and tested a class of finite difference methods for incompressible viscous flows which can be used in a variety of flow situations. These methods have been applied to flows of practical significance [Stri88] and extended for use with domain decomposition [Stri89]. Generation of vortices by an array of cylinders is an example of the type of flow to be computed by these methods. Vortex shedding by closely spaced cylinders is an important consideration in the design of heat exchangers, cooling tower arrays, and offshore structures, see e.g., [Lam88]. These flows are computationally difficult because small time steps are needed to resolve the vortex behavior, yet many time steps are required to obtain the long term behavior. We require systems with supercomputer capabilities to efficiently solve such problems.

[Glow88] Glowinski, R., G.H. Golub, G.A. Meurant, and J. Periaux, eds. **Proc. First Intern. Symp. on Domain Decomposition Methods for Partial Differential Equations**, SIAM, Phila. PA (1988).

[Lam88] Lam, K. and W. C. Cheung, "Phenomena of vortex shedding and flow interference of three cylinders in different equilateral arrangements", *Journal of Fluid Mechanics*, **196**, pp. 1-26 (1988).

[Stri88] Strikwerda, J.C. and Y.M. Nagel, "A numerical method for the incompressible Navier-Stokes equations in three-dimensional cylindrical geometry", *Journal of Computational Physics* **78**, pp. 64-78 (1988).

[Stri89] Strikwerda, J.C. and C.D. Scarbnick, "A domain decomposition method for incompressible flow" *Computer Sciences Tech Report 896*, University of Wisconsin-Madison (December 1989).

F.5.2.2. Efficient Solution of Elliptic Equations

This research is devoted to theoretical and experimental studies aimed at achieving effective, rapid solution of the large systems of algebraic equations which arise in numerical solution of boundary value problems for elliptic partial differential equations. When the operators are positive definite, or at least definite, there are many rather good methods. In particular, there are the very effective Multigrid methods. When the problems are indefinite, one is forced to use more complicated methods, e.g., a preconditioning strategy followed by a general iteration scheme, say the Conjugate Gradient method applied to the normal equations. These very indefinite problems arise in many applications. For example much of the acoustic work for determining geological structure, (e.g., in oil exploration) is based on the exterior indefinite Helmholtz equation.

Parter, his co-workers, and his students have studied Multigrid methods [Deck88], [Mand90], [Kamo86], [Kamo87] and preconditioning strategies [MaPa90], [Joub90], [Gold90]. This work has been both theoretical and experimental. Among these is a basic early work on the parallel implementation of Multigrid Methods vs SOR Methods [Kamo86], [Kamo87].

Much of the current interest in Domain Decomposition methods for elliptic problems stems from a fact which was very apparent in this work. While on a theoretical basis Multigrid methods are among the most effective for definite problems, the nature of certain Multigrid approaches causes the parallel implementation of these methods to degrade rapidly as the number of processors gets large. Indeed, seven or eight processors is already a large number in such cases. On the other hand, Domain Decomposition methods can make use of larger number of processors, such as would be available on our iPSC/X. Thus, while their theoretical rates of convergence may be inferior, their effective rates of convergence appear to be superior. We need the iPSC/X to do experimental studies of the convergence properties of these methods on large-scale problems.

[Deck88] Decker, N., Mandel, J., and Parter, S. V., "On the Role of Regularity in Multigrid Methods", in **Multigrid Methods, Theory, Applications, and Supercomputing**, McCormick, S.F., ed., Marcel Dekker, Inc. (1988), pp. 143-155.

[Gold90] Goldstein, C. I., Manteuffel, T. A., and Parter, S. V., "Preconditioning and Boundary Conditions without H_2 estimates: L_2 Condition Numbers and the Distribution of the Singular Values", *Los Alamos National Laboratory Report LA-UR-90-1856* (1990).

[Joub90] Joubert, W. D., Manteuffel, T. A., Parter, S. V., and Wong, S. P., "Preconditioning Second-Order Elliptic Operators: Experiment and Theory", *Los Alamos National Laboratory Report LA-UR-90-1615* (1990).

[Kamo86] Kamowitz D., "Theoretical and Experimental Results for a Variety of Multigrid Algorithms", *Ph.D. Thesis*, Computer Sciences Department, Univeristy of Wisconsin-Madison (1986).

[Kamo87] Kamowitz, D., "SOR and MGR[v] experiments on the Crystal Multicomputer", *Parallel Computing* 4(1987), pp. 117-142.

[Mand90] Mandel, J., and Parter, S. V., "On the Multigrid F-Cycle", *Applied Math and Comp.* 37(1990), pp. 19-36.

[MaPa90] Manteuffel, T. A., and Parter, S. V., "Preconditioning and Boundary Conditions", *SIAM J. Num. Anal.* (1990), pp. 656-694.

F.6. RESEARCH IN COMPUTER ARCHITECTURE

In many cases, new ideas in computer architecture are initially developed informally and at a high level of abstraction. More detailed analysis must then be done to characterize whether and where the idea should be used. In the following sub-sections, we discuss our current and proposed research on tools for evaluating future parallel architectures.

F.6.1. Simulation Testbed for Future Parallel Architectures (M. Hill, G. Sohi, M. Vernon, D. Wood)

For two reasons, designing a simulator of a multiprocessor is more challenging than designing a uniprocessor simulator. First, a multiprocessor simulator should allow the multiprocessor environment (e.g., memory system interactions) to affect the system's workload, obviating, in general, the use of traces and making (at least partial) workload re-execution necessary. Second, workload re-execution requires considerable processing power and

memory. We estimate simulating a 100-processor system on a non-trivial workload requires a simulation platform with at least 50 processors and a gigabyte of memory to avoid a slowdown of more than 1000 times.

For these reasons we plan to build a parallel architecture simulation platform to run on a iPSC/X. We will simulate large systems using Chandy-Misra techniques [Misr86]. Simulation performance will be improved by using approximations to reduce the frequency of synchronization in the simulations platform. An example approximation would be to allow the simulation of a processor in a cache-coherent MIMD to get all invalidations at cache miss time rather than asynchronously. The error caused by this approximation will be assessed by running some slow, precise simulations. Three uses for the proposed simulation platform are currently envisioned:

- *Scalable Synchronization Primitives.* This evaluation will assess the effect of request combining on the selection of synchronization primitives and hardware cache coherence mechanism in arbitrary interconnects, such as those permitted by the proposed IEEE Scalable Coherent Interface Protocol (SCIP) standard.
- *A Hybrid Shared-Memory Message-Passing Memory System.* This memory system will (1) manage coherence in hardware, (2) assist the compiler in managing coherence, and (3) allow a restricted form of message passing.
- *Exploiting Data Parallelism.* This research will investigate how problems with data parallelism are mapped onto a high-performance vector processor or a massively-parallel multiprocessor built from low-performance processors, to see what an appropriate "middle ground" parallel processing system might be.

A prototype of this simulator currently runs as multiple Unix processes on a workstation communicating via sockets. Current workloads must consist of independent programs compiled for a DECstation. We are in the process of allowing the simulated processes to share memory. After this is completed later this year, we will port the simulator to the department Sequent Symmetry and then next year onto an Intel iPSC/2. While adequate for debugging the prototype simulator, the above machines all lack the processing power and memory needed to make the simulator broadly useful. For this, we need to port our simulator to the iPSC/X requested in this proposal.

[Misr86] Misra, J., "Distributed Discrete Event Simulation", *ACM Computing Surveys* **18**, 1, pp. 39-65 (March 1986).

F.6.2. Prototyping New Multiprocessor Architectures (D. Wood, M. D. Hill, and M. K. Vernon)

A prototype implementation is the final step in evaluating new computer architecture ideas, conducted only after exhausting analytic models and detailed simulations. Analytic models require high-level abstractions which entail assumptions that are often difficult or impossible to completely validate. Detailed simulations of multiprocessors require (at least partial) workload re-execution, and therefore are too slow to fully simulate the large-scale applications typically run on these machines. Thus, to fully analyze many architectural ideas requires the detail and performance provided by building proof-of-concept prototypes.

We consider the iPSC/X as a unique opportunity to develop a large-scale MIMD multiprocessor prototype. Intel has offered to make available the network interface specification, and to provide technical advice in the development of prototype nodes. We envision adding new processor nodes, or replacing existing ones, that support both traditional iPSC/X message-passing and a shared-memory model, with caches and distributed main memory. By building the cache and memory controllers using field-programmable gate arrays, these nodes will provide the flexibility and instrumentation to experiment with a wide range of memory system architectures. Possible uses for such a flexible prototype environment are:

- *A Hybrid Shared-Memory Message-Passing Architecture.* Evaluating large applications on a hybrid system (see the previous section) requires a prototype implementation.
- *Evaluating Directory-Based Cache Coherence Protocols.* The flexibility of field-programmable gate arrays will permit the system to support a variety of coherence protocols, ranging from Agarwal's Dir_iNB to the IEEE Scalable Coherent Interface Protocol's linked-list protocols.

Building a large-scale multiprocessor prototype from scratch is very expensive and requires years of work. The iPSC/X provides the framework to quickly develop a flexible prototype system at a fraction of the normal cost in time and money.

F.6.3. Parallelizing Trace-Driven Cache Simulations (G. Sohi and M. D. Hill)

Well-designed cache memories are essential to the design of high-performance processors (both uniprocessors and multiprocessors) today. To evaluate the cache design space thoroughly, trace-driven simulation is the methodology of choice of the computer architect. To get accurate results with a trace-driven simulation, the number of references in a trace must be much larger than the cache size. For cache sizes of a few megabytes, one needs to simulate traces that contain billions of references. Trace-driven cache simulation for a single cache

organization has traditionally been considered a serial process. We have recently developed algorithms to carry out the simulation of a single cache organization in parallel. The main idea behind these algorithms is to partition the trace into several pieces, simulate each piece in parallel, and then merge the information from the individual pieces (similar research is also being carried out by Heidelberger and Stone [HeSt90].) Our algorithms appear well-suited to arbitrary machine paradigms. Preliminary experience with a Sequent Symmetry has suggested near-linear speedups in the simulation time. We will implement and test out the algorithms on the iPSC/X and the MasPar, with the overall goal of using available parallel hardware resources to reduce the simulation time needed to evaluate alternate cache organizations.

[HeSt90] Heidelberger, P. and H. S. Stone, "Parallel Trace-Driven Cache Simulation by Time Partitioning", *Proc. of the 1990 Winter Simulation Conference.*

F.7. RESEARCH IN ARTIFICIAL INTELLIGENCE

Section F.7.1 proposes to design and test parallel algorithms on the MasPar for analyzing dynamic sequences of images and for model-based 3D object recognition. Section F.7.2 proposes to develop algorithms on the MasPar and iPSC/X that perform a hierarchical sequence of image and feature transformations for recognition tasks. Section F.7.3 proposes extensions to hybrid learning methods that integrate explanation-based learning with neural network learning. Parallel implementations on the MasPar will be studied in order to thoroughly evaluate the performance of these methods.

F.7.1. Dynamic, Model-Based Computer Vision (C. R. Dyer)

As a monocular observer moves in a 3D world, the parts that can be seen and their geometry change with the vantage point. Very little is known about how to represent and use this information in computer vision. We plan to develop data-parallel algorithms on the MasPar for analyzing sequences of images in order to recover information about egomotion and object motion, and to recognize 3D objects. One specific goal is to quantitatively describe and model the relationships between motion and the change in the 2D appearance of a 3D shape. Because motion is such an integral part of this information processing process, a second major goal is to use a uniform, spatiotemporal approach to 3D object representation and to intermediate-level motion description for dynamic, model-based 3D object recognition.

To explicitly model the dynamical appearance of 3D objects as the viewpoint moves, we introduced "aspect space", which is the cross-product of the image plane and viewpoint space [Plan90]. Using this multidimensional space, we are developing new 3D object representations that explicitly describe the dynamic evolution of visible features over viewpoint. One of these

is called the rim appearance representation; it describes the exact appearance of the occluding contour of a shape as a function of viewpoint [Seal90]. Using this representation we are defining procedures for extracting and organizing viewpoint-dependent features of self-occlusion, e.g., T-junctions and other contour "events." In particular, we are currently developing methods for organizing features such as T-junctions so that they can be represented and used for dynamic, model-based recognition of 3D objects. We plan to design parallel indexing and matching algorithms for comparing dynamic features extracted from a spatiotemporal image sequence with the dynamic model features. Initial approaches to be implemented on the MasPar will be based on the data-parallel operation of matching each dynamic image feature with each dynamic model feature to determine the transformation (i.e., viewpoint) that maps one into the other. Each pair votes for its transformation, and the transformations with the greatest number of votes determine the possible solutions. This approach is a generalization of the Hough transform and has been used previously for 2D object recognition using static features on the Connection Machine [Tuck88].

We are also deriving new methods for computing intermediate-level motion descriptions directly from the motion of features in spatiotemporal (image plane $\times$ time) image-sequence volumes prior to object recognition and description. From this data, spatiotemporal surface flow can be computed [Allm90b], flow lines detected, and spatiotemporal surfaces segmented. Intermediate-level motion description such as cyclic motion [Allm90a] and T-junction motion can then be computed. Because of the massive amounts of data and floating-point computations involved in processing spatiotemporal image volumes, this work will use the MasPar so that algorithm design and testing can be done on long image sequences.

[Allm90a] Allmen, M. and C. Dyer, "Cyclic Motion Detection using Spatiotemporal Surfaces and Curves", *Proc. 10th Int. Conf. Pattern Recognition*, Atlantic City, N.J., pp. 365-370 (1990).

[Allm90b] Allmen, M. and C. Dyer, "Computing Spatiotemporal Surface Flow", *Proc. 3rd Int. Conf. Computer Vision*, Osaka, Japan, pp. 47-50 (1990).

[Plan90] Plantinga, H. and C. Dyer, "Visibility, Occlusion, and the Aspect Graph", *Int. J. Computer Vision* **5**, pp. 137-160 (1990).

[Seal90] Seales, B. and C. Dyer, "Modeling the Rim Appearance", *Proc. 3rd Int. Conf. Computer Vision*, Osaka, Japan, pp. 698-701 (1990).

[Tuck88] Tucker, L., Feynman, C. and D. Fritzsche, "Object Recognition using the Connection Machine", *Proc. Computer Vision and Pattern Recognition Conf.*, Ann Arbor, Mich., pp. 871-878 (1988).

F.7.2. Hierarchical Computer Vision (L. Uhr)

We have been developing parallel computer vision systems that are scalable, so they can be embedded into successively larger networks of processors and recognize moving objects in real time [LiUh87]. They input information into a large sensor array, and this information is

transformed by a commensurately large array of processors. Each processor receives information from a small subset of cells in the sensor array (usually this is the sub-array lying directly below it). Subsequent layers of processors then examine and transform that information until recognition is completed.

In its simplest form, this builds a tree of arrays, giving a pyramid- or cone-like structure [LiUh85, HoUh89]. However, a number of other structures are possible, and we are investigating what appear to be the most attractive, including multi-apex structures and re-cycling nets [Uhr 87]. One criterion for evaluating these structures is how well models of objects to be recognized can be embedded.

The MasPar's SIMD array can handle the large arrays of data with great speed. But the convergence and divergence of information moving through trees needs new shuffling algorithms. The iPSC/X is also likely to be reasonably efficient for most array and pyramid operations. Algorithms will be developed for both the MasPar and the iPSC/X so that their performance can be compared. In addition, a recognition system will be developed that uses the MasPar to process the very large arrays near the input image and the iPSC/X to process the more complex transformed information at higher levels. These will all be compared with one another, and with implementations on conventional serial computers of both our highly parallel-hierarchical algorithms and more traditional serial model-matching computer vision systems.

[HoUh89] Honavar, V. and L. Uhr, "Recognition Cones: A Neuronal Architecture for Perception and Learning", *Connection Science* **2**(1990,

[LiUh87] Li, Z. N. and L. Uhr, "Pyramid vision using key features to integrate image-driven bottom-up and model-driven top-down processes", *IEEE Trans. Systems, Man and Cyber.* **16**, pp. 250-262 (1987).

[LiUh85] Li, Z. N. and L. Uhr, "Comparative Timings for a Neuron Recognition Program on Serial and Pyramid Computers", *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, pp. 99-106 (1985).

[Uhr87] Uhr, L. "Highly parallel, hierarchical, recognition cone perceptual structures", in **Parallel Computer Vision**, L. Uhr, ed., Academic Press, New York, NY, pp. 249-292 (1987).

F.7.3. Hybrid Learning Methods for Large-Scale Problems (J. W. Shavlik)

We are developing a hybrid system that combines the symbolically-oriented explanation-based learning paradigm with the neural backpropagation learning algorithm. This system is currently being applied to large-scale, real-world problems such as recognition of important biological features in DNA sequences, prediction of the secondary structure of proteins, and recognition of handwritten characters. Due to the computational demands of neural network learning and the large numbers of training examples for these tasks, a parallel implementation of our learning algorithm is essential.

Briefly, our learning algorithm allows one to express, in a convenient verbal fashion, "preconceived notions" about the task being learned. These inferences need not be perfectly correct. Rather, they should be viewed as hints, heuristic guesses, and rules of thumb. Our algorithms map this information into a neural network, specifying the network topology and initial weight settings, and then use a collection of training examples to refine the original task-specific knowledge [Shav89, Towe90].

In conjunction with molecular biologists on campus, we are applying our machine learning algorithms to computational biology problems in the Human Genome Project. Prof. Blatter's research group in the UW Genetics Department is sequencing the DNA of the bacterium *E. coli*; we are using our algorithms to predict the locations of the genes in the DNA sequences his laboratory produces.

We propose to develop a parallel implementation of our learning algorithm for the MasPar machine based on the techniques presented in [Sing90]. This research promises to extend machine learning so that it applies to important scientific problems where there exist both roughly-correct theories of the task and large numbers of training examples. For example, we currently have encoded the Chou-Fasman algorithm [Chou78] for protein-structure prediction in a collection of over 200 inference rules and have collected over 20,000 examples. We are now using the examples to improve the accuracy of the Chou-Fasman algorithm. Due to the magnitude of data being produced in the Human Genome Project, parallel methods are greatly needed for this and many related tasks.

[Chou78] Chou, P. Y. and G. D. Fasman, "Prediction of the Secondary Structure of Proteins from their Amino Acid Sequence", *Advances in Enzymology* **45**, pp. 45-148 (1978).

[Shav89] Shavlik, J. W. and G. G. Towell, "An Approach to Combining Explanation-Based and Neural Learning Algorithms", *Connection Science* **1**, 3, pp. 233-255 (1989).

[Sing90] Singer, A., "Implementations of Artificial Neural Networks on the Connection Machine", to appear in *Parallel Computing*, (1990). (Also appears as Thinking Machine Corporation Technical Report RL90-2.)

[Towe90] Towell, G. G., J. W. Shavlik, and M. O. Noordewier, "Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks", *Proc. of the Nat. Conf. on AI*, Boston, pp. 861-866 (July 1990).

F.8. RESEARCH IN MODELS OF PARALLEL COMPUTATION

Section F.8.1 proposes to use the iPSC/X and MasPar systems to develop and test the validity of better conceptual models for predicting parallel computing performance. Section F.8.2 proposes to use the machines to develop and test better theoretical models for parallel algorithm design.

F.8.1. Conceptual Models of High Performance Parallel Computing (M. K. Vernon)

Broad questions that need to be answered before parallel computing can become the dominant method for solving large-scale applications include: 1) what computational model is most effective for harnessing parallelism? (e.g., SIMD, MIMD, loosely-SIMD, or ?), 2) how should the processors, memory modules, and I/O devices be interconnected to maximize processing power? and 3) how should the resources be allocated to multiple parallel applications that may request service from the system simultaneously? To address these questions quantitatively, conceptual models of computation-intensive applications are needed. Parameters that characterize real parallel workloads are also needed to use models to address questions 2 and 3 more completely than has been possible to date. The proposed facility will allow us to make significant progress in these areas.

One challenge is to develop workload models that are independent of the precise architecture the applications might execute on. This would allow us to evaluate a future architecture before it is available for workload measurements. Fortunately, there are applications, particularly in computational science, that provide the principal motivation for future high-performance parallel architectures and that appear to have definable computational needs. The applications with which we are familiar have a high degree of regularity and "data parallelism", but vary in their efficiency for SIMD execution and in their communication and I/O needs. We intend to work with various researchers, including the scientific and numerical applications group and the parallel optimization group, to characterize parallel applications along these dimensions.

We hypothesize that the model(s) that will ultimately be successful will characterize both the algorithm and its resource demands, and thus will include ideas from initial performance models (e.g., [EaZL89, VeJS88]) as well as from evolving models of parallel computation (see section F.8.2). We intend to use the iPSC/X and MasPar to test the validity of such conceptual models for predicting performance and to measure parameters of real workloads. As part of this work, we plan to compare the performance of SIMD and MIMD algorithms for important classes of large-scale applications, to analyze various interconnection structures for processors, memory, and I/O devices, and to analyze resource allocation policies for large-scale parallel system workloads.

[EaZL89] Eager, D. L., J. Zahorjan, and E. D. Lazowska, "Speedup vs Efficiency in Parallel Systems", *IEEE Trans. on Computers*, Vol. 38, No. 3, March 1989.

[VeLZ88] Vernon, M. K., E. D. Lazowska, and J. Zahorjan, "An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols," *15th Annual Int'l. Symp. on Computer Architecture*, Honolulu, Hawaii, May30-June 2, 1988.

[VeJS88] Vernon, M. K., R. Jog, and G. Sohi, "Analysis of Hierarchical Cache-Coherent Multiprocessors", *Performance Evaluation*, Vol. 9, No. 4, August 1989.

F.8.2. Toward a useful theory of parallel computation (E. Bach, A. Condon, P. Tiwari)

There is a pressing need for a theory of parallel algorithms that enables one to design and implement useful algorithms on real parallel machines. Indeed, the theory community is now in the curious situation of having identified a class of parallel algorithms (NC algorithms) that are, in many cases, inferior to ordinary sequential algorithms on existing parallel machines. We propose to address this problem.

The reason for this problem seems to be the following. For sequential algorithms, it has long been agreed that a rough (big-O style) time analysis suffices to identify promising algorithms from among the many contenders. In coming to this agreement, though, researchers could rely on many years of experience in computer design, algorithm development, and implementation, all of which helped to build a finely-honed intuition about efficient sequential computation. In contrast, no such consensus exists for parallel algorithms. Furthermore, we do not yet have this experience, particularly in implementation, to draw on in the parallel world, and there is no reason to expect that present-day models of parallelism should be complete. (It is well to remember that the original ''sequential'' model of computation was the Turing machine.)

We propose to evaluate and improve recently proposed models of parallel computation, e.g. the asynchronous-PRAM of Gibbons [Gibb89] and the BPRAM of Aggarwal, Chandra, and Snir [ACS89]. We will complement this with a detailed study of parallel algorithms for combinatorial and algebraic problems. In the process, we will implement algorithms for several problems on both the MasPar and the IPSC/X. We expect that the invaluable experience gained in this process will help us identify better models for studying parallel algorithms on parallel machines.

As a preliminary step in our program, we taught an advanced graduate class where we and students implemented a variety of algorithms on the department's 20-processor Sequent. Although this architecture represents just one approach to parallel computation, one can make the following general comments:

1. For many combinatorial problems (e.g. finding maximal independent sets), finding an implementation that beats a sequential solution is a challenging task. For others, it does not yet seem possible.
2. Even for a relatively small number of processors (10-20), communication and synchronization costs may far outweigh the ''real'' computation done by an algorithm. After the

algorithms were implemented, studies by T. Tsuei [Tsue90] confirmed that the limiting factor in many implementations was contention for a common processor-memory communication channel, not processor speed.

3. If one tries to use "off the shelf" PRAM algorithms of the type surveyed by Karp and Ramachandran [Karp88], the number of processors that can be fruitfully used is very small, e.g., on the order of a half-dozen.

It is possible, of course, that such algorithm misbehavior is peculiar to bus-based computers such as the Sequent, and newer architectures may allow processors to be better utilized. For this reason, we plan to implement existing parallel algorithms on the iPSC/X and MasPar. Among the target algorithms for implementation, we consider four to be particularly interesting: (1) A parallel version of a new GCD algorithm that (in serial mode) runs up to twice as fast as the conventional "binary" method [Sor90]. (2) The parallelizable algorithm by Ben-Or and Tiwari [Ben89] for approximating roots of a polynomial. (3) Parallel algorithms for basic operations on graphs, such as the construction of a breadth-first search tree, a depth-first search tree, or an ear-decomposition. (4) Algorithms to construct or detect a perfect matching in a graph; one possibility here is to use algorithms based on reducing matching to matrix calculations [MVV87].

We foresee two possible outcomes of this research. Ideally, we would identify a small set of architecture-independent "critical resources" that function much as time does in the study of sequential algorithms, allowing one to rank alternative proposed algorithms according to their practicality. However, it is also within the realm of possibility that no such set exists. Then, the task would be to build not one, but several, theories of parallel computation, each suited to designing good algorithms for a particular class of machines.

[ACS89] Aggarwal, A., Chandra, A. K. and M. Snir, "On Communication Latency in PRAM Computations", *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 11-21 (1989).

[Ben89] Ben-Or, M. and P. Tiwari, "Simple Algorithms for Approximating All Roots of a Polynomial with Real Roots", *invited to J. of Complexity*. pp. 1989.

[Gibb89] P. B. Gibbons, "A More Practical PRAM Model", *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 158-168 (1989).

[Karp88] Karp, R. and V. Ramachandran, "A Survey of Parallel Algorithms for Shared-Memory Machines", *Technical Report No. UCB/CSD 88/408*, University of California–Berkeley (March 1988).

[MVV87] Mulmuley, K., Vazirani, U. V. and Vazirani, V. V., "Matching is as Easy as Matrix Inversion", *Combinatorica* 7, 1, pp. 105-113 (1987).

[Sor90] J. Sorenson, "The k-ary GCD Algorithm", *In preparation*, University of Wisconsin–Madison.

[Tsue90] Tsuei, T. "Speedup Prediction and Diagnosis for Shared Memory Multiprocessor Systems", *Ph.D. Dissertation*, University of Wisconsin–Madison (To appear in 1990).

H. SUMMARY OF PREVIOUS ACCOMPLISHMENTS

In this section we very briefly describe the main results that were obtained using either the 20-processor Sequent S-81 multiprocessor or the 32-processor Intel iPSC/2 hypercube acquired with funds from the Topaz grant. In addition to these two machines, funds from the Topaz grant helped purchase workstations for faculty members and graduate students in the department. To take advantage of the total computing power that such a collection of machines provides, we developed a batch scheduling facility known as Condor. This facility has been widely used and has made it possible to obtain a large number of results that could not have been obtained otherwise. Due to space considerations, however, these results are not included in the following sections.

H.1. The Gamma Multiprocessor Database Machine (D. J. DeWitt)

The focus of the Gamma project [DeWi90] has been the design and evaluation of high performance parallel database systems. Gamma employs three key technical ideas that facilitate scaling the system to very large configurations. First, all relations are declustered across multiple disk drives, enabling relations to be scanned in parallel. Second, novel parallel algorithms based on hashing are used to implement the complex relational operators such as join and aggregate functions. Third, dataflow scheduling techniques are used to coordinate multi-operator queries.

The Gamma prototype on the iPSC/2 has been used for a variety of projects. [Schn89] examined the performance of four different parallel join algorithms. A study of several different strategies for processing complex relational queries in a parallel database system can be found in [Schn90]. A multiuser performance evaluation of Gamma's three alternative declustering strategies [Ghan90a] indicated that no single strategy was always superior and lead to the development of a new-declustering strategy which is described and evaluated in [Ghan90b]. Finally, we have designed a new high-availability strategy, termed chained declustering [Hsia90], to ensure the availability of all data in the event of processor and/or disk failures.

H.2. Parallel, Real Time Vision (C. Dyer)

Given a dense temporal sequence of images of a known three-dimensional object which is moving arbitrarily, we designed several new real-time, parallel algorithms for object tracking [Verg90a, Verg90b]. The first used a model driven, hypothesize-and-test approach, with 3D-to-2D prediction and matching. The second used a 2D-to-3D matching method which dynamically tracked the motion of 2D edge contours from frame to frame. Both algorithms were implemented as coordinated parallel procedures on two connected multiprocessors, the Sequent S-81

and an 8-stage Aspex Pipe image processing computer, which are connected to one another through a Multibus interface. The Pipe has an attached CCD camera which digitizes 60 frames per second. We implemented the low-level image processing parts of the algorithms (e.g., edge detection) on the Pipe and the high-level 3D model storage and analysis parts on the Sequent. Pipelining was used on both machines to efficiently solve both partitioning of work into sub-tasks as well as applying these operations repeatedly over time on successive images so as to produce a continuous stream of results at the input frame rate.

We have also developed several new parallel algorithms for stereo vision that are based on the integration of multiple constraints, and are defined in terms of a connectionist model of computation [Stew88a, Stew88b, Stew90]. The results for both natural and synthetic images showed nearly linear speedups for a wide range of conditions and parameters.

H.3. Genetic Algorithms for Assembly Line Balancing (M. C. Ferris)

In order to assess the effectiveness of genetic algorithms for problems in combinatorial optimization, we implemented a parallel genetic algorithm for the assembly line balancing problem on the iPSC/2 hypercube using a local neighborhood scheme for mating [And90]. Not only does this parallel version obtain close to linear speed-up over the serial version, but the method also significantly outperforms the standard version in terms of the quality of solutions produced. The results demonstrate that parallel schemes of this sort are extremely effective at producing approximately optimal solutions with little knowledge of the problem domain, and therefore would be an invaluable tool for use by optimization consultants.

H.4. Condor - A Hunter of Idle Workstations (M. Litzkow and M. Livny)

The goal of the Condor system is to make it possible for users whose computational needs are not satisfied by a single workstation to take advantage of the large number of unused CPU cycles that are available in a pool of workstations such as ours [Mutk87]. Condor, by monitoring the load average, keyboard, and mouse activity on all of the machines in its ''worker pool,'' automatically detects idle workstations. Once an idle machine has been located, one of the jobs awaiting execution in Condor's ''batch'' queue is selected and initiated. Even though a Condor program executes on a remote workstation, all system calls are routed back to the submitting workstation. Thus, a Condor program sees the same execution environment as if it had been run on the submitting machine. Transparent mechanisms are provided for checkpointing and migrating a job as necessary. This allows Condor to remove a job from a workstation when its regular user returns, and gives Condor users assurance that their jobs will make progress and eventually complete. No modification of the user's program is necessary; it only needs to be linked with a

special Condor library.

Within the Department, Condor's worker pool consists of more than 200 heterogeneous workstations. In the last year, it has provided more than 6,000 days of CPU time to more than 150,000 jobs [Litz90]. At least ten researchers have each accumulated more than 200 days of CPU time during this period. Condor is available to the Internet Unix community via anonymous FTP and, to date, over two hundred sites have fetched the software.

H.5. Parallel Optimization Algorithms (O. L. Mangasarian)

Our principal aim has been to develop computationally efficient and mathematically sound parallel algorithms for the solution of a variety of optimization problems. In [Mang88], a gradient projection successive overrelaxation (GP-SOR) was proposed and tested for the solution of symmetric linear complementarity problems and linear programs. In [DeLe88a], serial and parallel successive overrelaxation (SOR) methods were proposed for the solution of an augmented Lagrangian formulation of the dual of a linear program. A parallel implementation on the Sequent yielded speedup efficiencies of over 65% for problem sizes of up to 10,000 constraints, 40,000 variables and 1,400,000 nonzero matrix elements. Convergence for asynchronous parallel SOR algorithms for the symmetric linear complementarity and the multi-sweep asynchronous parallel SOR algorithm for the nonsymmetric linear complementarity problem were established in [DeLe88b] and [DeLe90b], respectively. Computational tests on the Sequent demonstrated the superiority of the multi-sweep asynchronous SOR algorithm over the single-sweep version on both symmetric and nonsymmetric linear complementarity problems. In [DeLe90b], a new method of parallelizing Lemke's algorithm for solving linear complementarity problems was developed for the 32-node Intel iPSC/2, yielding speedup efficiencies as high as 76%. By combining the effects of concurrency and vectorization, the computing time in some cases was reduced by a factor of 100. In [Seti90], serial and parallel implementations of the interior dual proximal point algorithm for the solution of large linear programs were described and tested on the Sequent S-81, using a preconditioned conjugate gradient (PCG) method. Tests on a set of multicommodity network flow problems showed that the PCG method outperformed direct methods of solution.

H.6. Parallel Algorithms for Large-Scale Network Optimization (R.R. Meyer)

Parallel algorithms (using asynchronous variants of the network simplex method and a variety of decomposition tools) were developed for large-scale network optimization problems of various classes, including pure networks, generalized networks, and multicommodity networks. In the case of single-commodity networks, the algorithms that were devised [Clar90]

included several asynchronous variants of the network simplex method. These techniques took advantage of a speculative method that overlapped the typically consecutive pivoting and pricing phases, using parallelization of the pricing operations and a very quick recheck of the results of the pricing phase (which is weakly dependent on the data modified in the pivoting phase) prior to the next execution of the pivot. These asynchronous algorithms were implemented on the Sequent S-81 and efficiently solved (with good speedups) problems with more than 1,000,000 arcs. For multicommodity problems, a structured interior point method [Schu90] was developed. This method takes advantage of problem structure in several ways, including the construction of parallel network subproblems and the use of a multidimensional coordination phase. Solution times for the so-called Patient Distribution System test problems, now in common use for testing purposes, were at least an order of magnitude better than competing algorithms for this problem class.

H.7. Parallel Program Development (B. Miller)

Our debugging work has been based on a technique is called *flowback analysis* [Mill88b, Mill90a]. When flowback analysis is applied to parallel programs, it avoids the need to set breakpoints and re-execute the program [Mill88a]. Our research has been addressing the compile-time semantic analyses for flowback analysis debugging, execution-time trace generation, and debug-time resolution of dynamic dependences. We have designed and implemented a research prototype of our ideas for a subset of the C programming language (everything except pointers) on the Sequent. The prototype uses an interactive graphical interface that allows the programmer to select and follow dynamic execution dependences. We also reduce the cost of execution-time tracing by incrementally generating detailed traces as the user makes queries at debug-time [Choi90]. This project also includes support for automatically detecting data races in a parallel program's execution [Netz90].

A second aspect of our research has been the development of measurement and analysis tools for parallel programs running on both loosely- and tightly-coupled multiprocessors [Mill87, Yang89, Mill88c, Mill90b]. These tools not only provide detailed statistics, but also provide information to directly guide the programmer to the cause of performance problems. Critical Path Analysis [Yang88] is one such technique. A second technique is based on identifying phases in a program's execution, and then using this information to focus the programmer's attention on more detailed performance problems.

H.8. Searching Game Trees in Parallel (M. Solomon)

Game playing programs require a great deal of computation and would thus appear to be an ideal candidate for parallel implementation. In fact, current champion chess-playing programs generate alternative moves and evaluate board positions using parallel algorithms, but the central part of game planning—game-tree search—has thus far proved resistant to efficient parallel implementation. In order to achieve high efficiency, a parallel algorithm must perform *speculative work*—work that is performed before it is proved necessary. We have designed and implemented a new algorithm for the Sequent, $\mathcal{ER}$, that appears better suited to parallel implementation [Stei90]. For comparison, we also implemented one of the best previously described algorithms, "pv-splitting." For our experiments, we measured the performance of both algorithms on both synthetically-generated random game trees and on trees generated by a program that plays Othello. For sufficiently deep trees (7 to 11 plys), $\mathcal{ER}$ achieves more than twice the efficiency and, therefore, twice the speedup of the pv-splitting algorithm with 27 processors (based on experiments where we used a large machine that Sequent made available to us).

H.9. Parallel Program Speedup Analysis (M. K. Vernon)

We used the Sequent S-81 to investigate the use of analytical performance models to predict and diagnose parallel program speedups [TsVe90] by developing an efficient hierarchical model to predict speedup degradation due to three factors: 1) available parallelism and fork/join overhead, 2) lock contention, and 3) contention for hardware resources (i.e., the system bus and memories). The model is efficient because it can be solved in a strictly top-down manner with no need to iterate between the three levels, and because each level's submodel is efficient.

We showed that the overall speedup predicted by the model is quite accurate for several alternative parallelizations of the Gaussian elimination algorithm for solving simultaneous equations, and also for a program that implements Tarjan and Vishkin's algorithm for computing the biconnected components of an undirected graph [Tavi85]. In each case, the overall predicted speedups are within 10% of the measured speedups (and often much closer) [Tsue90].

The hardware contention model parameters are obtained using non-obtrusive hardware probes in the form of a Northwest Instruments 2000 logic analyzer. This monitoring system provided a prototype for a monitoring system with more storage capacity that we are currently developing for the Symmetry S-81 [VeHi90].

H.10. References

[And90] Anderson, E.J. and Ferris, M.C. "A genetic algorithm for the assembly line balancing problem", *Proceedings of the Integer Programming/Combinatorial Optimization* Conference", Waterloo, Ontario, Canada, May 28-30, 1990.

[Choi90] J.-D. Choi and B.P. Miller, "Code Generation and Separate Compilation in a Parallel Program Debugger", in **Languages and Compilers for Parallel Computing**, Research Monographs in Parallel and Distributed Computing, MIT Press and Pitman Publishing, D. Gelernter, A. Nicolau, and D. Padua, eds., 1990.

[Clar90] Clark, R. and R. Meyer, "Parallel Arc-Allocation Algorithms for Optimizing Generalized" Networks" , *Annals of Operations Research* , **22,** pp. 129-160 (1990).

[DeLe88a] De Leone R. & O. L. Mangasarian "Serial and parallel solution of large scale linear programs by augmented Lagrangian successive overrelaxation", *Lecture Notes in Economics and Mathematical Systems*, **304,** pp. 103-124, Springer-Verlag, Berlin (1988).

[DeLe88b] De Leone R. & O. L. Mangasarian "Asynchronous parallel successive overrelaxation for the symmetric linear complementarity problem", *Mathematical Programming, Series B*, **42,** pp. 347-361 (1988).

[DeLe90a] De Leone R., O. L. Mangasarian & T.-H. Shiau "Multi-sweep asynchronous parallel successive overrelaxation for the nonsymmetric linear complementarity problem", *Annals of Operations Research*, **22,**pp.

[DeLe90b] De Leone R. & T. T.-H. Ow "Parallel Implementation of Lemke's Algorithm on the Hypercube", *UW Computer Sciences Technical Report 905*, submitted to ORSA Journal on Computing (January 1990).

[DeWi90] DeWitt, D., Ghandeharizadeh, S.,Schneider, D., Hsiao, H., Bricker, A., and R. Rasmussen, "The GAMMA Database Machine Project," *IEEE Transactions on Knowledge and Data Engineering*, Vol 2. No. 1, March, 1990.

[Ghan90a] Ghandeharizadeh, S. and D. DeWitt, "A Multiuser Performance Analysis of Alternative Declustering Strategies for Selection Queries", *Proc. of the 6th International Conference on Data Engineering*, Los Angeles, CA, (February 1990).

[Ghan90b] Ghandeharizadeh, S. and D. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines", *Proc. of the 1990 VLDB Conference*, Brisbane, Australia, (August 1990).

[Hsia89] Hsiao, H. and and D. DeWitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines," *Proc. of the 6th International Conference on Data Engineering*, Los Angeles, CA, (February 1990).

[Litz88] Litzkow M., Livny, M., and Mutka M. W., "Condor - A Hunter of Idle Workstations" *Proc. of the 8th International Conference on Distributed Computing Systems,* San Jose, California, (June 1988).

[Litz90] Litzkow, M., and Livny, M., "Experience With the Condor Distributed Batch System", *IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, (Oct. 1990).

[Mang88] Mangasarian, O. L. & R. De Leone "Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems and linear programs", *Annals of Operations Research*, **14,** pp. 41-59 (1988).

[Mill87] B.P. Miller, C. Yang, "IPS: An Interactive and Automatic Performance Measurement Tools for Parallel and Distributed Programs", *Proc. of the 7th Int'l Conf. on Distributed Computing Systems*, West Berlin (September 1987).

[Mill88a] B.P. Miller, J.-D. Choi, "Breakpoints and Halting in Distributed Programs", Proc. of the 8th Int'l Conference on Distributed Computing Systems, San Jose, (June 1988).

[Mill88b] B.P. Miller, J.-D. Choi, "A Mechanism for Efficient Debugging of Parallel Programs", *Proc. of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, Atlanta (June 1988).

[Mill88c] B.P. Miller, "DPM: A Measurement System for Distributed Programs", *IEEE Trans. on Computers* 37, 2, pp. 243-247 (February 1988).

[Mill90a] R.H.B. Netzer and B.P. Miller, "Detecting Data Races in Parallel Program Executions", in **Languages and Compilers for Parallel Computing**, Research Monographs in Parallel and Distributed Computing, MIT Press and Pitman Publishing, D. Gelernter, D. Gross, A. Nicolau, and D. Padua, eds., 1990.

[Mill90b] B.P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, and S.-S. Lim, "IPS-2: The Second Generation of a Parallel Program Measurement System", *IEEE Trans. on Parallel and Distributed Computing* **1**, 2, pp. 206-217 (April 1990).

[Mutk87a] Mutka, M. W., and Livny, M., *Proc. of the 7th International Conference on Distributed Computing Systems*, Berlin, West Germany, (September 1987).

[Mutk87b] Mutka, M. W., and Livny, M., "Profiling Workstations' Available Capacity For Remote Execution" *Proc. of Performance-87, The 12th IFIP W.G. 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, Brussels, Belgium, (December 1987).

[Netz90] R. Netzer and B.P. Miller, "On the Complexity of Event Orderings for Shared-Memory Parallel Program Executions", *Proc. of the 1990 Int'l Conf. on Parallel Processing*, St Charles, Ill., August 1990.

[Schn89] Schneider, D. and D. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," *Proc. of the 1989 SIGMOD Conference*, Portland, Oregon, (May 1989).

[Schu90] Schultz , G. and R. Meyer, "A Structured Interior Point Method", *Computer Sciences Department Technical Report #934* , University of Wisconsin, Madison, WI (1990), submitted to proceedings of the Symposium on Parallel Optimization 2, Madison, Wi, July, 1990.

[Schn90] Schneider, D. and D. DeWitt, "Tradeoffs in Processing Multi-Way Join Queries via Hashing in Multiprocessor Database Machines", *Proc. of the 1990 VLDB Conference*, Brisbane, Australia, (August 1990).

[Seti90] Setiono, R. "Interior dual proximal point algorithm using preconditioned conjugate gradient", *UW Computer Sciences Technical Report 951*, submitted for publication (July 1990).

[Stei90] Steinberg, I. and M. Solomon, "Searching Game Trees in Parallel," Proceedings of the 1990 International Conference on Parallel Processing Aug. 13-17, 1990, pp. III-9 to III-17.

[Stew88a] Stewart, C. and C. Dyer, "Local Constraint Integration in a Connectionist Model of Stereo Vision", *Proc. Computer Vision and Pattern Recognition Conf.*, Ann Arbor, Mich., pp. 165-170 (June 1988).

[Stew88b] Stewart, C. and C. Dyer, "The trinocular general support algorithm: A three-camera stereo algorithm for overcoming binocular matching errors", *Proc. 2nd Int. Conf. on Computer Vision*, Tampa, Fl., pp. 134-138 (Dec. 1988).

[Stew90] Stewart, C. and C. Dyer, "Parallel Simulation of a Connectionist Stereo Algorithm on a Shared-Memory Multiprocessor", in **Parallel Algorithms for Machine Intelligence and Pattern Recognition**, V. Kumar, P. Gopalakrishnan and L. Kanal, eds., Springer-Verlag, New York, pp. 340-359 (1990).

[TaVi85] Tarjan, R. and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithm," SIAM J. Computing, vol. 14, no. 4, 1985, pp. 862-874.

[Tsue90] Tsuei, Thin-Fong, "Speedup Prediction and Diagnosis for Shared Memory Multiprocessor Systems", Ph.D. Dissertation, University of Wisconsin–Madison, October 1990, in preparation.

[TsVe90] Tsuei, T.-F. and M. K. Vernon, "Diagnosing Parallel Program Speedup Limitations Using Resource Contention Models", *Proc. Int'l. Conf. on Parallel Processing*,

[Verg90a] Verghese, G., Gale, K. and C. Dyer, "Real-Time Motion Tracking of Three-Dimensional Objects", *Proc. IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, pp. 1998-2003 (May 1990).

[Verg90b] Verghese, G., Gale, K. and C. Dyer, "Real-Time, Parallel Motion Tracking of Three-Dimensional Objects from Spatiotemporal Image Sequences", in **Parallel Algorithms for Machine Intelligence and Pattern Recognition**, V. Kumar, P. Gopalakrishnan and L. Kanal, eds., Springer-Verlag, New York, pp. 310-339 (1990).

[VeHi90] Vernon, M. K. and M. D. Hill, "A High-Speed Data Acquisition System for Research in Parallel Computing", NSF Grant No. CDA-8920777.

[Yang88] C. Yang, B.P. Miller, "Critical Path Analysis for the Execution of Parallel and Distributed Programs''', *Proc. of the 8th Int'l Conf. on Distributed Computing Systems*, San Jose (June 1988).

[Yang89] C. Yang, B.P. Miller, "Performance Measurement of Parallel and Distributed Programs: A Structured and Automatic Approach", *IEEE Trans. on Software Engineering* **15**, 12, pp. 1615-1629 (December 1989).