UVA Proposal to NSF Institutional Infrastructure Program

Anita K. Jones
James H. Aylor

Computer Science Department TR-90-15
July 1990

UVA Proposal
to
NSF Institutional Infrastructure Program

Anita K. Jones
James H. Aylor

Abstract

This technical report contains sections excerpted from the proposal which the departments of Computer Science and the Electrical Engineering submitted to the National Science Foundation Institutional Infrastructure Program in December 1989. That proposal was subsequently adapted somewhat and was awarded over $5.5M including matching funds from the Commonwealth of Virginia and the University of Virginia.

In this proposal we ask for support -- primarily equipment, students and staff -- to enhance the capabilitites that will permit the joint departments to do hardware and software experiments not now possible or feasible. We need the capability to perform experiments more rapidly, in more depth and with greater quality and fidelity. Experimental capabilities include the design of custom chips, integration of custom and off-the-shelf hardware into board systems, the creation of software systems and applications, the exercise and instrumentation of the prototype software on the prototype hardware. The collective faculty involved strongly believe that the quality of computer research can frequently be enhanced if experiments can be realized to help test the scientific hypothesis under investigation.

This work is being supported by Grant No. CDA-8922545 funded by the National Science Foundation.

B. Executive Summary

This proposal is jointly submitted by the faculty of the Department of Computer Science and the computer engineering faculty of the Department of Electrical Engineering of the University of Virginia. The infrastructure for which we ask NSF resources is vital to the experimentation that is the foundation of our multi-disciplinary research.

The University of Virginia granted its first doctoral degree in computer science in 1968. During the last five years, the Department has grown dramatically. Annual research funding for CS has increased from $300,000 to $2.4 million. During the past two years faculty quality has increased with the addition of three professors from Carnegie-Mellon and one each from Arizona, Illinois and Rochester. Facilities have improved from 8,000 sq. ft. to 18,000 sq. ft. renovated in 1989. Equipment has increased from a VAX 780 to a VAX 8600 and 43 Suns. Both the quality and number of graduate students have increased. In the past two years the size of the graduate class has grown from 76 to 127. This year's entering graduate class will have the highest average GRE scores in the School of Engineering (math plus verbal scores of 1348). It will be our largest class and will contain the largest number of entering Ph.D. students.

Research in parallel computing is performed within the Institute for Parallel Computation (IPC) which was created by the Computer Science faculty. The IPC houses 128-node and 32-node hypercubes and a 32-node BBN Butterfly. The IPC was established by computer science and systems engineering faculty. This is one example of the Departments' interdisciplinary research. Other work involves Biomedical Engineering, Biology, Electrical Engineering, and Applied Mathematics. New research liaisons are planned with the National Radio Astronomy Observatory (in Charlottesville) and the Nuclear Magnetic Resonance Institute in the UVa Medical School.

Research in computer engineering is conducted in the Center for Semicustom Integrated Systems (CSIS) within the Department of Electrical Engineering. Formed in 1984, the Center has grown to have annual research funding of $1.2 million with 4 faculty, 25 graduate students, and two technical staff members. Laboratory space of 1,716 sq. ft. is being renovated during 1989-1990. The Center currently operates 3 minicomputers, 8 Sun and 7 Apollo workstations, 4 AED 512 color graphics viewports and a Tektronix DAS 9100 tester. The Center has substantial industry support ($300,000 during CY89) from companies including Hughes, IBM, Mentor Graphics, and General Electric. CS and EE have a long-standing, close working relationship as evidenced by our joint Masters degree in Computer Engineering, cross-listed courses, and our on-going joint research projects.

The University and the School of Engineering have been very supportive of EE and CS. All space utilized by these departments has been renovated in the past two years. Space available to CS (EE) has been increased by more than 200% (80%) in that time. Architectural pre-planning for a new computing building will be complete in Fall 1989. Further support is evident in the matching resources for this proposal: $1.6 million.

The majority of faculty in the CS and EE departments perform experimental research. We simulate, specify, prototype, reconfigure, fabricate, instrument, and test hardware/software systems and components in order to validate our scientific and engineering hypotheses. Our supporting experimental infrastructure consists of laboratory space, equipment—hardware and software—and people. Most important are knowledgeable students, faculty and staff. They integrate the other elements to form a research environment in which experiments can be devised and implemented with an acceptably low investment of time and intellectual energy.

The desired experimental infrastructure provides a spectrum of related capabilities that support research in the four general categories shown in Figure B-1. We use the term *end-to-end* to suggest a sequence of implementation choices. A researcher may closely control whatever aspects are critical.

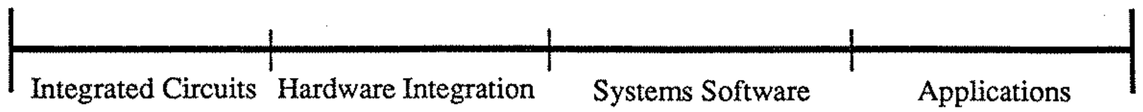

Figure B-1. Research Categories

At one end is the capability to create custom circuits. Further along the spectrum is the capability to integrate off-the-shelf, and perhaps custom, components to create a board system. Still further along is the capability to program small stand-alone software modules. They may execute on a custom board system or on a general computing engine. At the other end is the capability to experiment with both general systems and applications. Not all research investigations will cover the *end-to-end* spectrum. Each researcher will choose the needed subset of capabilities. Such an infrastructure offers the researcher the option of performing experiments at precisely the most effective points of the hardware/software experimental space.

Equipment to support integrated circuits is in a laboratory that is located in the Center for Semicustom Integrated Systems (CSIS). This proposal requests hardware and software to enhance that laboratory. Infrastructure for hardware integration and low-level software instrumentation will be located in the Systems Integration Laboratory (SIL), which is to be built. Equipment for both systems software and applications is mainly in the form of general-purpose computing engines and software. The networked computing resources of CS, IPC and CSIS will be upgraded with resources requested in this proposal. Of particular importance are the resources that will improve student access and computing capacity.

We perform research in the following areas:

Computer architecture (Aylor, Davidson, Grimshaw, Johnson, Jones, Wulf)
Programming language design (Cook, Grimshaw, O'Bagy, Pratt, Wulf)
Compilation (Davidson, Wulf)
Fault tolerant hardware and software (Johnson, Knight)
Parallel/distributed simulation (Aylor, Reynolds)
Parallel, scientific, and object-oriented databases (French, Jones, Pfaltz, Son)
Test technology (Aylor, Johnson, Knight)
VLSI design technology (Aylor, Cohoon, Johnson, Williams)
Real-time and control systems (Cook, Weaver)
Vision systems (Olson, Martin)
Operating systems (Cook, Grimshaw, Jones, Wulf)
Networks (Weaver)
User interfaces (Pausch)
Parallel genetic algorithms (Cohoon, Martin, Richards)
Computational geometry (Cohoon, Martin, Richards, Salowe)

We have a substantial record of research results. We want to increase the experimental component of our research. This requires bringing together the necessary knowledge, tools, skills, and processes. We want to decrease the intellectual effort and the elapsed time to accomplish a desired experiment. An improved infrastructure will lead to more experiments, more experimental data, and more analysis. Therefore, scientific results will be better substantiated, more quickly produced and of better quality.

Compared to other computer science and engineering departments, we are a focused *systems* group with a strong experimental bias. There is a synergy to be gained because of this focus: some research groups produce hardware, software or processes that can be experimental support for other projects. To give one example, the compiler project can rapidly retarget a compiler. This process is useful to a machine architecture project refining an Instruction Set Processor definition. It is also useful for a real-time operating system project wishing to port to alternative target architectures. We are at the point that we can take advantage of such synergy. The requested resources will permit us to package selected research output. Internal use of the tools and processes we build is the first step toward national distribution.

Note: Bill Wulf is the AT&T professor in the School of Engineering and Applied Science. He is currently on leave as Assistant Director of Computer & Information Science & Engineering at the National Science Foundation. To avoid any appearance of conflict of interest, he has not been involved in the preparation of this proposal. His vita does appear because he is the principal investigator on the DARPA-funded WM machine architecture project. He will return to UVa in mid-1990.

Summary of the Research to be Supported

The description of our research activities appears in five sections, each of which closely relates to one of the research categories in Figure B-1. Two sections, Real-Time Systems and Parallel Systems, are in the Software Systems category. The following paragraphs summarize Section F (Research).

VLSI Design Automation and Validation: Several projects have the objective of creating an integrated environment in which a digital designer can create VLSI circuits and systems efficiently and effectively. Projects include physical design and modeling.

Physical Design: VLSI physical design is a multi-step process whose goal is to translate a logical description of a digital system into a physical package. The basic steps are partitioning, floor-planning and placement, and routing. Over the past three years we have developed isolated tools for each basic step. They are objectively judged to be among the best of the state-of-the-art. The tools depend upon non-traditional techniques such as geometric search techniques using computational geometry and genetic algorithms. One of these tools, BEAVER, is able to route each of the classic switchbox instances in less than two seconds. In addition, both wire usage and via usage are better than or comparable to the best previously known solution. The future objective is to design and prototype selected tools and then to integrate them into a physical design environment together with tools of commercial and research origin. The infrastructure is crucial to being able to build an integrated tool set in contrast to isolated tools.

Uninterpreted/Interpreted Modeling: Analysis and validation of digital system designs is accomplished via modeling. Unfortunately the many tools available each focus on a particular phase of design; different modeling tools exist for the system level, behavioral level, logic level and circuit level. The designer must translate between the different tools. As a result, costs increase along with errors and design time. Our goal is to develop a design representation, and associated tools, suitable for initial conceptualization and refinement to final physical implementation. Initially, a design is represented in an uninterpreted model in which tokens representing information whose form and meaning is unknown flow among elements whose function is not well specified. As the design is refined, elements are described in an interpreted model whose function, inputs, and outputs are specified. Research questions derive from the coexistence of uninterpreted and interpreted models and "mixed-model" simulation.

Computer Architecture: Computer architectures and the interface between hardware and software vary rapidly. We discuss three projects in this area.

The WM Machine Architecture: The WM project is exploring a RISC-like architecture that enables concurrent execution of several instructions. In principle, 13 RISC-equivalent instructions can be executed per cycle; in practice 4-5 instructions per cycle seem the norm for real applications. This means that by using the WM architecture one can design a processor with several times the speed of a traditional RISC processor using the same technology. We have built an instruction set processor simulation for the execution of benchmarks; results are encouraging. We have designed and fabricated several WM IC chips. Future WM research requires the ability to integrate custom and industrial components into systems capable of executing experimental operating systems.

Retargetable Compilers: Compilers are the critical tool for exploiting new machine architectures as they emerge. Our long term research goal is to develop compiler technology that enables a computer scientist to construct a production-quality compiler for a new architecture with less than one month's effort. We have developed a technology for retargeting optimizing compilers and built compilers for C, (validated) Ada, and Pascal and have retargeted compilers to 12 machines. Future research will involve: 1) implementation of a more powerful machine description language and 2) optimization at link-time and at post-execution time.

Fault Tolerance and Testing: Both testability and fault tolerance must be considered from the start of a digital design. Our research focuses on an *active* design method which requires elimination of system faults via reconfiguration. Such a methodology depends upon detecting component faults concurrent with normal operation. We are experimenting with three alternative concurrent fault detection techniques. Based on the best of these, we will populate a library of fault-tolerant building blocks. The design tools will be taught the fault-tolerant characteristics of the building blocks so that it is possible to perform reliability analyses on new systems designs.

Real-time Software: We study operating systems, databases and communications designed to serve real-time applications. We implement them so are concerned with software fault tolerance.

The StarLite development environment: We have developed a host environment, StarLite, for the development of real-time operating systems and real-time databases. StarLite is unusual in that it provides for the execution of target software on virtual single, multiple and distributed processors. Interfaces to a variety of devices, such as Ethernet, exist. Hence StarLite is conducive to the development of software for which hardware does not exist or is inhospitable. StarLite executes on Unix for portability and includes an initial set of development tools including debugger, visualizer and a software library of over 200 modules. Our complementary objectives are to mature the StarLite environment and validate its use for real-time systems.

Real-Time Communications: We have built a variety of LAN networks for ships, aircraft and factories and have achieved very high speed transmission rates. Now we are moving to higher speed networks to focus on the new research problems that surface. First, we are developing LAN protocols for control systems where latency control is critical to meeting real-time deadlines. Traditional transport protocols are optimized for throughput; the needed latency control is absent. Second, we are studying dynamic manipulation of message importance and the role of pre-emption as a means to assure real-time performance. Third, we will address protocol compilation and validation for the situations in which real-time communications protocols will be tailored to the specific system needs.

Software Fault Tolerance: Most approaches to software fault tolerance attempt to deal with all faults. In contrast, we are investigating several strategies which will support tolerance of specified classes of faults. One technique relies on the fact that many computer programs map multiple inputs to one output. For some applications it is possible that an alternative input will cause the software to traverse a different path and produce the originally desired correct output. Such techniques take advantage of knowledge of the limited faults of interest.

Parallel Systems: A wide variety of parallel systems research is pursued at UVa. Three projects are described in some depth.

ADAMS, a parallel database interface: The goal of the ADAMS—the Advanced Data Management System—project is to create a standard interface which will support parallel access and management of data. It will 1) interface directly with programs written in algorithmic languages, particularly scientific programs, 2) obviate the need for a separate database system, and 3) provide for parallel access to data. Research problems include determining data description and access primitives, naming vast amounts of data, organizing data on parallel storage devices for efficient access, and ensuring integrity.

Parallel simulation: Our research in parallel simulation has resulted in the development of SPECTRUM, a testbed for discrete event simulation. Within SPECTRUM it is possible to vary the protocol which determines how the concurrent, cooperating simulation processes interact. The testbed environment is useful for experimentation with alternative protocols and the development of new protocols. In addition, it permits simulation of operators which provide parallel processes efficient, lockless access to data structures. We have proposed such operators both for simulation and general purpose computing.

Parallel genetic algorithms: In a genetic algorithm a population of solutions evolves through successive generations. Recently, we have devised a new genetic algorithm that is suitable for distributed implementation. In parallel, multiple populations of solutions evolve, each in a separate environment. Their value is judged by a fitness function. At intervals, fit solutions in one environment are transplanted to another. We have shown that the performance of the distributed genetic algorithm exceeds that of the original genetic algorithm.

Applications: We discuss two applications.

Computer vision: Our work is in the area of *dynamic scene analysis*. We emphasize the control of *focus of attention* mechanisms in vision systems in a parallel processing context. We want to understand the fundamental nature of quasi-parallel agents cooperating to interpret visual information in dynamic environments. Second, we seek to define effective operations on multiple, varying-grain resolutions of time-sequenced images.

Computational geometry toolbox: Computational geometry is concerned with manipulating, processing and examining geometric objects. We have selected a set of geometry problem areas: convex hulls, Voronoi diagrams and Steiner trees. These are of particular use in VLSI design layout and in computer vision. We will build a library of algorithms to solve problems in these geometry areas. Our emphasis will be on data structures that are common to multiple algorithms, simplicity of control and data structures, robustness of algorithms to input perturbations and dynamic insertions and deletions. We will then experiment with suitability and ease of use of the toolbox algorithms in VLSI design layout and computer vision software.

F. Research

We want to expand our infrastructure for the support of experimental systems research. The infrastructure to do so involves people, equipment, space, and ideas. It is characterized as *end-to-end* because the capabilities facilitate experimentation at one end with the design of circuits, all the way to instrumentation and test of prototype complex software systems at the other end. Because the infrastructure is "what is needed", precise definition is difficult. The infrastructure is best illustrated by examples of the capabilities it offers:

- Construct prototype compilers for experimental target processors.
- Construct prototype operating, real-time and control systems by replacement of modules within existing systems.
- Compose and test prototype boards from custom and off-the-shelf components.
- Design, layout, simulate, and test VLSI circuits, with routine fabrication performed at a national (MOSIS) or industrial facility.
- Store, manipulate, incrementally alter and re-simulate experimental designs at many levels.
- Model and analyze systems using queueing and petri net models; simulate components and systems at all levels.
- Instrument, monitor and test both hardware and software, with automatic test generation wherever possible.
- Produce technical prose and presentation visuals — both still (transparencies) and animated graphics (video).

No single researcher may necessarily use all the capabilities *end-to-end*. Each user will instead select the set of capabilities needed. The presence of infrastructure to support a slightly broader experiment will encourage researchers to attempt more experimentation.

Much of our research is characterized as "systems". We build many prototype hardware and software components. We build software that is sensitive to underlying hardware, e.g. user interfaces, networks, vision, operating systems, compilers, and databases.

Because of our systems focus and our productivity in the past, several research projects have built or are building software, tools, or hardware that other research groups wish to use. The list of such opportunities includes the vpo retargetable compilers, StarLite real-time operating system kernel, SPECTRUM simulation testbed, BEAVER and LIR VLSI routers, and the fault tolerant chip library. This opportunity for synergy offers high leverage for extending the experimentation of some researchers. With an infrastructure enrichment grant from NSF we will commit to packaging some of the research products for use by others at UVa. It is a straightforward, though not simple, step beyond that to distribute to other research sites. We aspire to be a national asset in this regard.

We divide the infrastructure intuitively into four areas, each offering a set of capabilities of the kind listed above. The four areas are depicted by thick lines in Figure F-1 below. The general areas in which we do research are indicated by thin lines. Placement and length of a solid research line indicates what portion of the infrastructure serves our research in that area today. Dotted research lines are placed to illustrate how an enriched infrastructure will broaden the kind of experimentation that is performed. An arrow indicates that some research project in that area is expected to produce a tool or a process that itself will be refined so that it becomes a part of the infrastructure.

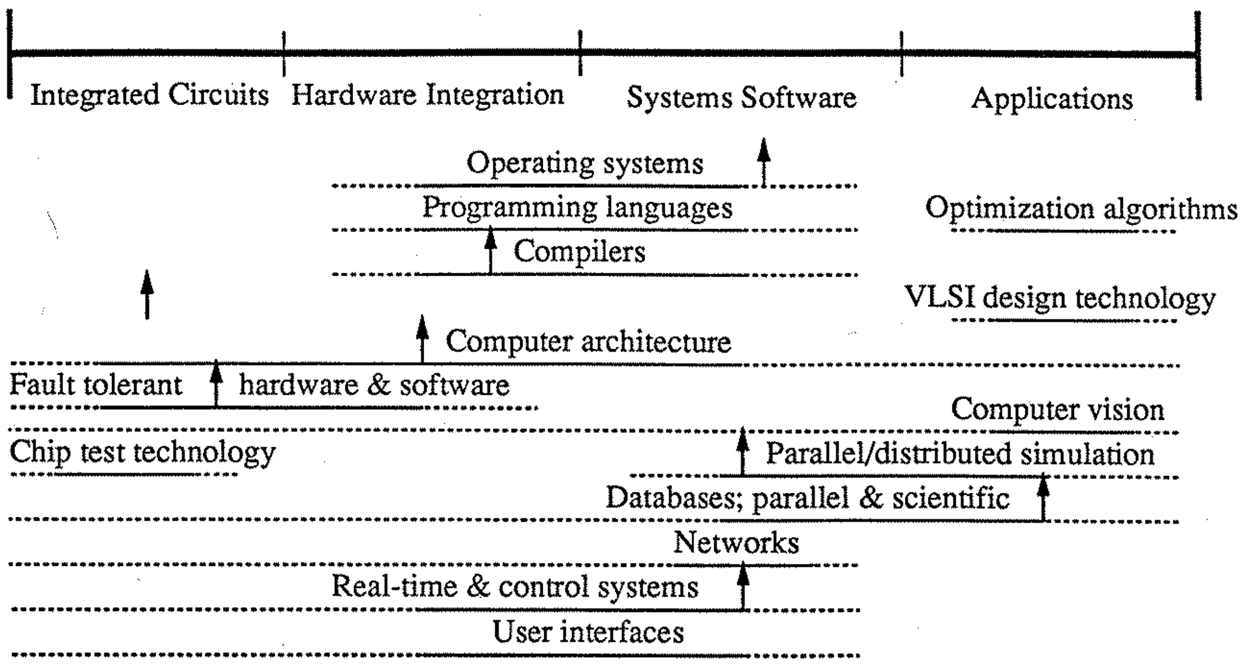

Figure F-1. Research Spectrum

Our plan to enrich our infrastructure would proceed as follows. First in 1990, we will strengthen the general computing environment as well as the integrated circuit design facility which is physically part of the Center for Semicustom Integrated Systems. We would acquire workstations, laboratory equipment and software tools from commercial sources and other research sites. We would also hire technical staff and graduate students to integrate the new facilities and to ensure their consistency and ease of use for researchers.

Second in 1991, we will create the Systems Integration Laboratory (SIL) so that we have the ability to integrate hardware systems. The SIL will permit the circuit builders to insert custom circuits in board-level systems. At the same time it will permit those researchers who now build software that is sensitive to the hardware interface to vary that interface in experiments. During 1991 we will begin to package the expertise, tools and technology we have in six areas: networking interface software, operating and real-time system kernel software, retargetable compilers, physical design tools, hardware component libraries, and the parallel simulation testbed. These packages will be made available to research projects other than the creating project. Projects that will benefit from developments in the above seven areas are as follows:

Capability	**Using Research Areas**
network interface	fault tolerant distributed architectures
OS/real-time system software	WM machine architecture, reliable control systems of physical systems (electric wheelchair design)
retargetable compilers	networks, real-time OS, WM machine architecture

physical design tools	all integrated circuit design activities
component libraries	all integrated circuit design activities
parallel simulation testbed	functional simulation of ICs

Lastly, beginning in 1991 we will incrementally improve our infrastructure both with funds from this NSF grant, and with other funds that we expect this grant to catalyze. We will routinely add new tools, expertise and processes to our environment.

Research to be supported by the infrastructure is described in five sections. Each describes several related projects; collectively these projects span the research categories in Figure F-1. Two sections, Real-Time Software and Parallel Systems, relate to the Systems Software research category. NSF review guidelines led us to focus on those research projects with a track record. Projects just getting under way have shorter descriptions. For each research project we state its long term research goals, past success and its research agenda—the next research questions to be asked and the milestones to be accomplished.

F.1 VLSI Design Automation and Validation

Digital designers, who create state-of-the-art VLSI circuits and systems, need quality tools that quickly and automatically perform a variety of design and management functions. While such tools are important in the creation of most complex systems, they are especially important in the design and implementation of digital products because of the large implementation costs and short development times. Also, digital systems are increasingly present in systems where failure can result in endangering human life or large financial loss, so that safety and reliability are crucial. Through our interdisciplinary VLSI custom design facility, we have developed and are continuing to develop a collection of high quality tools and methodologies that span the design spectrum. Our past successes and the resultant interest they have generated are readily documented [CoH88, HAW89].

In our research programs, we have established particular expertise in several major design validation and automation areas: fault tolerant design [BJA88, Joh89], physical design [CoR88, CoH88], modeling and simulation [AuA86, HAW89], and testing [AJR86, FAC89]. Below we describe some of the design tools and methodologies we have developed for two of these areas—physical design and system modeling.

F.1.1 Physical Design (Cohoon)

Physical design is a multi-step process whose goal is to translate a logical description of a digital system into a physical package. Its basic steps are partitioning, floor-planning and placement, and routing [PrL88]. The *partitioning* step decomposes the system into a collection of circuits, each of which can be realized as a single physical entity. The *floor-planning and placement* step determines where the circuit elements or *modules* that make up a circuit should be located and how they should be positioned. Finally, the *routing* step realizes the interconnections that comprise a circuit.

Our objective is to design and prototype selected tools, and then to integrate them into a physical design environment together with tools of commercial or other origin. The physical design project began three years ago. It has involved 5 faculty and 9 graduate students from 2 departments, 8 funding awards and has resulted in 15 journal and conference publications as well as isolated tools for each of the basic steps. They are objectively judged to be among the best of the state-of-the-art [CoR88, CoH88].

High performance is always a figure of merit for VLSI design tools because of the rapid growth in the size and complexity of circuits. We have repeatedly applied and extended techniques from non-traditional areas to solve problems yet attain high performance. For example, our partitioning and routing tools [Coh86, CoR88, CoH88, CoL90] depend upon generalized algorithmic and search techniques from computational geometry [PrS85]. Our floor-planning and placement tools [CoP87, CHM88] employ an adapted genetic algorithm method [Gol88, Hol75] to produce sequential and parallel combinatorial optimization schemes and depend upon an interesting generalization of the now traditional VLSI optimization method of simulated annealing [KGV83, SeS85].

Below we describe briefly two of our recent routing tools—LIR and BEAVER. These two tools demonstrate our ability to achieve research goals.

LIR—Detailed Router The routing of a circuit is decomposed into a collection of wiring sub-problems. Appropriate problem-specific routers are then invoked on the sub-problems. LIR is a special purpose tool for the most ubiquitous of the routing sub-problems—interconnecting two terminals in the presence of obstacles. Although traditional design environments provide either a maze router or a heuristic line search router [Sou81], both methods have major disadvantages in running time or failure to route.

We produced an alternative method called line intersection routing (LIR) [CoR88]. It uses a variation of a computational geometry technique, *line sweeping*, to pre-process the circuit, constructing a more suitable graph structure. This structure is then examined using efficient, special-purpose, rectilinear versions of shortest-path search algorithms. They exploit some computational-geometry-based theorems that we proved regarding the nature of optimal routings. The result is the first algorithms with polynomial running times in the number of obstacles that can determine optimal wirings with respect to the minimum length, minimum bend, and the minimum length-bend weight metrics. Thus our method offers an arbitrary worst-case speed-up over conventional methods. In addition, our routing experiments demonstrated that the method is equally effective in practice [CoR88].

BEAVER—Switchbox Router In the switchbox problem a collection of *terminals* are located on the perimeter of a rectangular region that may contain obstacles. All terminals belonging to the same signal or *net* must be interconnected within the switchbox by the router. The switchbox routing problem is one of the most difficult routing sub-problems [Sou81]; until recently no router could claim 100% routabilty. This limitation has caused some top VLSI chip manufacturers to delay using methodologies that introduce switchboxes. Recently, Joobbani and Siewiorek developed a knowledge-based expert-system router, WEAVER, that attains 100% routability [JoS86]. However, there was a price extracted for WEAVER's consistent success—many of its solutions even for relatively small instances required tens of minutes of CPU time.

We have developed a fast heuristic switchbox router called BEAVER. As with LIR, it uses computational geometry techniques. The figure of merit for BEAVER was both quickness and quality of route. The investigation was successful in both regards. BEAVER was able to route each of the classic switchbox instances in less than two seconds. In addition, for each of these instances, BEAVER's feedthrough usage is better than the best previously known solution and its wire usage is either better than, or comparable to, the best previously known solution.

The ideas validated in LIR and BEAVER are currently being generalized in our next router DRAGNET, a general purpose router for multi-layer circuits. Besides handling arbitrarily-shaped rectilinear routing regions with embedded obstacles, DRAGNET will avoid traditional, but unnecessary assumptions. For example, DRAGNET does not use a reserve layer strategy for horizontal and vertical wire segments. In fact, DRAGNET uses non-rectilinear wire segments (eg., 45° wire) if they better optimize the routing figure of merit.

The infrastructure enrichment is a necessary basis for us to be able to integrate isolated tools into an environment which will necessarily include industrial and university VLSI layout software components that interface with our tools. We need the advanced program development environment, as well as the staff to provide programming continuity, to perform integration, and to support the physical design environment when used in the chip laboratory. The several faculty and graduate students actively examining VLSI problems need convenient access to color workstations and printers. Today within the Computer Science department, we have access to a single color workstation and no color printers.

F.1.2 Uninterpreted/Interpreted Modeling of Digital Systems (Aylor, Johnson, Williams)

The advent of computer aided design has produced many tools and methodologies for the design, analysis and verification of digital systems. Unfortunately, each method has focused on a particular type of system or phase of design, so that supporting tools are very specific. As a design proceeds through refinement of abstract concept into detailed design, it passes through many levels during which analyses and simulations are applied. Each analysis and simulation is based on a different model expressed in a different language and tool. For example, there are different languages for modeling at the system level, behavioral level, logic level, and circuit level.

Because of this fragmented approach, the designer must translate the design between different tools. As a result, costs increase along with errors and design time. Our research goal is to develop a design methodology and associated tool(s) that represent a digital design from initial concept to final physical implementation. The project commenced in 1984 and now involves 3 faculty, 6 graduate students, and to date has published 3 papers. Interest is quite high in this area and we have recently received grants from Hughes, IBM and the Semiconductor Research Corporation.

We distinguish between two kinds of models. At the early design stage are *uninterpreted models* in which "tokens" that represent information—whose form or meaning is not known—flow among elements whose function is not well specified. In an uninterpreted simulation an element acts only on the presence or absence of a token. In contrast, a late design stage model is described as an *interpreted* model. Here information flowing in the model has a well-defined form and a known value. Elements act upon the value presented and have known input to output mappings.

We are investigating hybrid models in which uninterpreted elements coexist with interpreted elements. Communications between these different types take place through interfaces which convert tokens to values or values to tokens.

Today, environments support simulation for each of the two extreme models, but there is no automated support environment for hybrid models. Uninterpreted modeling is well researched. This work generally falls into two categories, petri nets [Pet81] and queuing models [Lav83]. Queuing models work well for gaining statistical data on very high-level uninterpreted models of digital systems. But when these models are further partitioned, it has proved to be difficult to express the deterministic nature of the system [Aul87]. In addition, models based on these techniques are often very large and cumbersome.

The only commercially-available tool that attempts to provide both types of modeling is the Architecture Design and Assessment System (ADAS) developed by the Research Triangle Institute [FSC85, Res87]. The ADAS tool supports hierarchical design with each element defined as a new model. Interpreted modeling is provided by allowing the user to write Ada code fragments that describe the function of an element. ADAS also provides functional simulation at lower levels of abstraction by allowing conversion of the ADAS models to a hardware description language form.

Our approach is to start with existing hardware description languages (HDLs). HDLs and their simulators accurately and conveniently represent the physical implementation, i.e. interpreted models, of digital systems at the switch, gate, register-transfer, and behavioral levels. By adding an uninterpreted modeling capability to hardware description languages based on extended petri nets and queuing models [AuA86], a *single design environment* can span design concept to implementation [AuA86, Aul87, HAW89]. At early stages uninterpreted modeling of a design would be used. Later, as individual elements are developed or components are selected from existing libraries, the system description would be systematically converted into a fully interpreted description for final verification. Such an approach, if successful, would bring uninterpreted modeling into the mainstream of the digital system design process utilizing modern hardware description languages.

We have selected VHDL, VHSIC Hardware Description Language as our basis language[IEE88]. We believe that VHDL will handle uninterpreted modeling so that analysis and simulation will pose no major problems. The difficult research questions include: What are the characteristics of the elements in an uninterpreted model that are the basis for performing system level estimation of performance, reliability, element utilization and throughput? Is the behavior of estimations well behaved as elements are incrementally described in an interpreted fashion? For example, it is not difficult to collect statistics such as the percent utilization of each element. As elements represention are refined to an interpreted form, do the statistics refine accurately?

How is the mapping from token to value defined in a hybrid model? What is the space of interactions between interpreted and uninterpreted elements? For example, if the interpreted element performing reconfiguration control for a fault-tolerant system emits a value which is translated to a token, what behavior is exhibited by uninterpreted elements receiving that token? More generally speaking, what are the feedback modalities between the two types of elements? Can an automated code generator be built to generate VHDL code for high level models described in a notation less detailed than VHDL?

Our long term goal is to answer such questions and to enrich a VHDL environment to support uninterpreted modeling resulting in the single design environment, and then to demonstrate its efficacy in practice.

References

[AuA86] R. J. Auletta and J. H. Aylor, A Non-Interpreted Model for Digital System Design, *Proceedings of the 1986 IEEE Southeastcon*, Richmond, VA, March 1986, 296-299.

[Aul87] R. J. Auletta, *An Uninterpreted Model for Hardware Description Languages*, Dept. of Electical Engineering, University of Virginia, 1987.

[AJR86] J. H. Aylor, B. W. Johnson and B. J. Rector, Structured Design for Testability in Semicustom VLSI, *IEEE Micro 6*,1 (February 1986), 51-58.

[BJA88] L. A. Belfore, B. W. Johnson and J. H. Aylor, The Design of Inherently Fault-Tolerant Systems, in *Concurrent Computations -- Algorithms, Architecture, and Technology*, Plenum Press, New York, NY, 1988.

[Coh86] J. P. Cohoon, Fast Channel Graph Construction, *Congressus Numerantium*, Vol. 53, 1986.

[CoP87] J. P. Cohoon and W. D. Paris, Genetic Placement, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems CAD-6*,6 (November 1987), 956-964.

[CoR88] J. P. Cohoon and D. S. Richards, Optimal Two-Terminal α-β Routing, *Integration: The VLSI Journal*, February 1988.

[CoH88] J. P. Cohoon and P. L. Heck, Beaver: A Computational-Geometry-Based Tool for Switchbox Routing, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems CAD-7*,6 (June 1988), 684-697.

[CHM88] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. S. Richards, Floorplan Design Using Distributed Genetic Algorithms, *IEEE International Conference on Computer-Aided Design*, Santa Clara, November 1988, 452-455.

[CoL90] J. P. Cohoon and S. C. Losen, A Priority-Queue-based Algorithm for Circuit Partitioning, *Integration: The VLSI Journal*, to appear in 1990.

[FAC89] E. L. Feldhousen, J. H. Aylor, J. P. Cohoon and B. W. Johnson, The Use of Genetic Algorithms in the Compaction of Randomly Generated Test Sets, *IEEE Transactions on Computer Aided Design of VLSI Circuits*, 1989. Under review.

[FSC85] G. A. Frank, C. U. Smith and J. L. Cuadrado, An Architecture Design and Assesment System for Software/Hardware Codesign, *Proceedings of the 22nd IEEE/ACM Design Automation Conference*, June 1985, 417-424.

[Gol88] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1988.

[HAW89] F. T. Hady, J. H. Aylor, R. D. William and R. Waxman, Uninterpreted Modeling using the VHSIC Hardware Description Language, *Proceedings of the 1989 IEEE Conference on Computer-Aided Design (ICCAD-89)*, Santa Clara, CA, to appear in November 1989.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[IEE88] IEEE, Inc., IEEE Standard VHDL Language Reference Manual, 1988.

[Joh89] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, Reading, MA, 1989.

[JoS86] R. Joobbani and D. P. Siewiorek, Weaver: A Knowleged-Based Routing Expert, *IEEE Design & Test 3*,1 (February 1986), 12-23.

[KGV83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, *Science 220*,4598 (May 13, 1983), 671-680.

[Lav83] S. S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press Inc., New York, NY, 1983.

[Pet81] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1981.

[PrL88] B. T. Preas and M. J. Lorenzetti, eds., *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, CA, 1988.

[PrS85] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[Res87] Research Triangle Institute, *Architecture Design and Assessment System User Manual Version 2.4*, Research Triangle Park, NC, 1987.

[SeS85] C. Sechen and A. Sangiovanni-Vincentelli, The TimberWolf Placement and Routing Package, *IEEE Journal of Solid-State Circuits SC-20*,2 (1985), 510-522.

[Sou81] J. Soukup, Circuit Layout, *Proceedings of the IEEE 69*,10 (October 1981), 1281-1304.

F.2 Computer Architecture

Computer architectures and the interface between hardware and software are varying rapidly. Dynamic change is driven by low-level electronics and communications technologies, and to a lesser degree by the software technologies in which the systems and application functions are rendered. To explore improved architectures requires investigation and experimentation with hardware implementation techniques, operating systems, compilers, and languages—in a cyclic feedback fashion. Several projects at UVa explore such architectural issues.

F.2.1 WM Machine Architecture (Wulf, Jones, Aylor, Davidson, Johnson, Grimshaw)

The WM project is exploring a RISC-like architecture that enables concurrent execution of several instructions. In principle, 13 RISC-equivalent instructions can be issued per cycle; in practice, 4-5 instructions per cycle seem the norm for real applications. This means that using the WM architecture one can design a processor with several times the speed of a traditional RISC processor using the same technology. WM is especially suitable for embedded and multi-computer applications. Specifically, it:

- has a peak performance 13 times that of RISC architectures for comparable gate-counts (area) and implementation technology.
- is capable of vector-like performance on computationally intensive numeric computations.
- can achieve vector performance on loops containing recurrences.
- does not require "heroic" compiler technology in order to achieve its peak performance.
- has data paths which have been carefully designed so that modest pinouts are adequate.

A key architectural feature that facilitates several of these properties is that selected WM registers are first-in/first-out register sequences that permit *streaming* of data, condition codes and instructions.

In addition to the CPU innovations, the memory system and IO system were designed so that user-level (applications) programs can access and control IO devices directly without compromising security or invoking operating system overhead. The same mechanism permits direct user-level access to communication and multi-computer synchronization devices.

The WM project began in late 1988 and involves 6 faculty and 6 graduate students. During 1989 the Instruction Set Processor (ISP) is being analyzed and refined [Wul88, WuH89, Wul89]. Complementary activities involve benchmark measurement of increasingly large and complex codes, C compiler development, and experimental component design and fabrication. Two models exist. A high-level ISP model (10,000 lines of C running on Sun, Mac II, and VAX 8600) is used for benchmark code and implementation timing experimentation. Although benchmarks have a synthetic flavor because actual WM processors do not yet exist, benchmark results compare favorably to reported statistics for the MIPS and Intel 860. A VHDL model (5,000 lines) exists to support component design efforts. Implementation experience is being gained from the design and fabrication of component circuits. Several WM IC chips were designed by students in the VLSI systems class, have been fabricated at MOSIS (Metal Oxide Semiconductor Implementation Service), and are now under test. The first chip for the Stream Control Unit (control) has been fabricated at MOSIS in 2 micron technology and is under test in the integrated circuit design laboratory.

Using the retargetable compiler technology described in the next section, a C cross-compiler exists; it has been shown to be robust enough to compile Unix utility code, which is the next set of benchmarking we will perform. Efforts in code optimization are under way. Next an Ada compiler will be retargeted. Coding is in progress for a bare-bones initial operating system to explore the utility of the WM system features for device access, protection and virtual memory.

The next stage of WM research will focus on systems—rather than components—performance and delivery of the performance to applications. This involves construction of several medium to high performance versions of WM processors, for example an embedded WM processor with 64-bit float capability. Construction of a high performance processor will involve an industrial partnership with a company having a suitable design and fabrication capability. In parallel we at UVa will continue with complementary component design. We will use MOSIS technology to implement a slower speed implementation, for example, a multi-computer WM implementation. The WM project will need the Systems Integration Laboratory to integrate custom and industrial components into systems capable of executing experimental operating systems.

With actual hardware implementations available, we can experiment with larger scale operating systems: general purpose, real-time, and control. The C retargetable compiler will translate the StarLite real-time kernel for experimentation when such board systems are built. Memory interconnection and very wide paths between processors and memories, and between processors and i/o devices will be explored. The WM project is one of the projects that has use for all of the "end-to-end" infrastructure capabilities.

The final stage of the project will be to realize a few application systems that deliver something near the maximum performance achievable with the architecture and implementation technology.

F.2.2 Retargetable Compilers (Davidson)

Compilers are the critical tool for exploiting new machine architectures as they emerge. Our long term research goal is to develop compiler technology that enables a typical computer scientist to construct a production-quality compiler for a new architecture with less than one month's effort.

We have developed a technology for retargeting compilers using vpo, the Very Portable Optimizer [BeD88, Dav86]. For compiler experts, our compilers have proven to be easily and quickly retargeted; yet they produce excellent code. Over the next years we plan to 1) reduce the time for a ''non-expert'' to retarget a compiler, 2) add additional optimization capability to both compilers and linkers, 3) validate the quality of code produced against excellent hand-coded compilers, and 4) package the compiler prototyping technology so that it can be routinely used at UVa and possibly at other research sites. This research currently involves one faculty and six graduate students and work has been documented in over 12 published papers and technical reports.

The optimizer, vpo, replaces the traditional code generator used in many compilers. The optimizer is retargeted by supplying a description of the target machine. Using the diagrammatic notation of Wulf [WJW75], Figure F-2 shows the overall structure of a set of compilers constructed using vpo. Vertical columns within a box represent logical phases which operate serially. Columns divided horizontally into rows indicate that the subphases of the column may be executed in an arbitrary order. IL is the Intermediate Language. Register transfers or register transfer lists (RTLs) describe the effect of machine instructions and have the form of conventional expressions and assignments over the hardware's storage cells. Any particular

RTL is machine specific, but the *form* of the RTL is machine independent.

Currently, we support the compilation of four source languages: C, Ada, Modula-2, and Pascal. The Ada compiler is front-end is also used in a validated commercial compiler. Both the C and the Pascal compilers pass validation suites that are in wide use in the industry to test compilers. The Modula-2 compiler is in process.

The vpo C compiler has been retargeted to twelve machines: VAX-11, Intel 386, Motorola 68020, AT&T DSP32, Intergraph Clipper, National Semiconductor 31016, Sun SPARC, IBM RT/PC, Harris HCX-9, Concurrent Computer Corporation 3200, AT&T 3B15, and WM. It has been ported to ten of these machines†. In all cases, the vpo-based compiler generates code that runs at least as fast as the code produced by the C compilers supplied with the machine. In most cases, the vpo-based compiler generates substantially faster code. This compiler technology has been licensed to AT&T Microelectronics, Intel Corporation, and Concurrent Computer

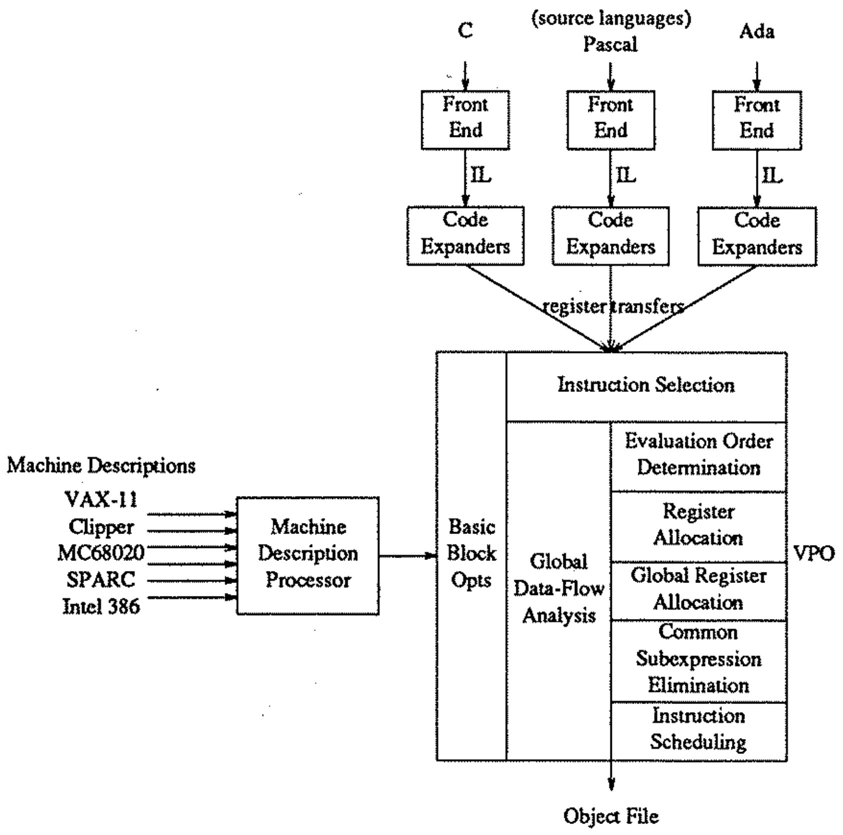

Figure F-2. Compiler Structure

†The DSP32 and WM C compilers run on VAXes and Suns.

Corporation for use in developing high-performance compilers to support chips and machines they market.

To achieve our long-term goal of making high-quality compiler technology available to the typical computer scientist, we are starting two new activities. First, we will develop a new machine description language. Currently, machine descriptions are written using *yacc*. Early on *yacc* had advantages. It is widely available and compiler writers are familiar with the peculiarities of writing grammars for *yacc*.

While the use of *yacc* has been successful, it is not ideal. The most important problem is that some compiler-relevant aspects of machines are impossible to describe using *yacc*. To schedule instructions so as to avoid pipeline conflicts or bubbles, the compiler must know pipeline characteristics. Currently, vpo depends upon a procedural description of a machine's pipeline which is separate from the machine description. *Yacc* descriptions also do not permit description of instructions that contain implicit loops—e.g., block move instructions. Lacking such a description, the optimizer cannot discover sequences of primitive instructions that can be replaced by the more complex loop instruction. Our objective is to develop a notation and the supporting tools that permit us to retain the general structure of our optimizer yet succinctly, simply and efficiently write and process machine descriptions.

Second, we will optimize at link-time and post-execution time. We have developed a prototype optimizing linker [BeD88]. It performs inter-procedural optimization of specific call and parameter usage by the callee, based on seeing all routines, regardless of the compile unit in which they were processed. We will extend the linker to perform global register allocation (i.e., allocation of heavily used global variables to registers), removal of dead functions, better local register allocation by using information in the static call graph, and some inlining.

Post-execution optimization is based on an efficient profiling of programs. The profiler yields detailed—down to the instruction level—information about the execution behavior of programs at an average increase of ten per cent of the execution time of the programs. To perform post-execution optimization we collect profiling information with the program running with representative data. The profiler counts variable references, loop iterations, and procedure invocations. This information guides the register allocation algorithms and inlining algorithms. Both the profiler and the optimizer exist; next we will integrate them and experiment with their effectiveness.

F.2.3 Architectural Support for Fault Tolerance and Testing (Johnson, Aylor)

The importance of testability escalates with the complexity of hardware. The importance of dependability escalates with the ubiquity of function. Architectural features to enhance both testability and dependability must be an integral component of design. Both depend upon detecting errors at a low level. The advent of Very Large Scale Integration (VLSI) technology has made the implementation of error detection and fault handling practical because of size, weight, power, and cost reductions. At the same time, the increasing complexity of today's systems has made the incorporation of fault tolerance and testability a daunting design problem.

Fault tolerance can be accomplished through the use of the so-called *active* approach. It requires that faults be detected, located, and eliminated from the system via reconfiguration. The cornerstone of the active approach is a fault detection ability. Therefore, techniques must be incorporated into the processing elements of a system to support fault detection and methodologies must be developed that use these elements to implement fault-tolerant systems. To be time-efficient, the fault detection technique used must operate concurrently with the normal function of the element.

While some research has been performed [Dri83, RAE81, SeL80], neither a good design methodology nor basic building blocks that employ concurrent, on-line fault detection yet exist. Our goal is to devise a methodology. We will then develop the computer-aided design tools, complemented by a library of standard building blocks, to reduce the methodology to practice. These standard blocks, such as adders or memories, will have on-line error detection and testability characteristics known to the tools and will be suitable for incorporation into larger, fault-tolerant, testable systems. Ideally, the method, tools, and building blocks can be used by those not proficient in fault tolerance techniques. This research was initiated in 1984 and involves 2 faculty and 4 graduate students. Results to date, some of which are sketched below, are documented in 10 papers and 3 technical reports, as well as a book [Joh89].

The two fundamental problems are: find error detection techniques that permit design of suitable building blocks for logic circuits and develop the methodology with the concomitant tools. In what follows, we first discuss three concurrent error detection techniques under investigation. We conclude with a discussion of the methodology issues.

Two fundamental approaches to error detection are hardware redundancy and time redundancy. The hardware redundancy approach says replicate the hardware and compare results. The time redundancy approach says compute the function twice, serially, using slightly different logic paths and compare the results. We have developed a hybrid technique called REcomputing using Duplication With Comparison (REDWC) [JAH88]. REDWC takes advantage of the speed of hardware redundancy and the hardware savings of time redundancy. Time redundancy can be effectively used because of the serial nature of the functions considered. The REDWC approach was originally applied to the design of a VLSI adder; the result used approximately 33% less gates and 37% less time than a purely time redundant adder. It required approximately 44% less gates and 7% more time than a pure hardware redundant adder. It appears that the hybrid approach is better than either end of the spectrum, unless very high performance is a requirement. We have shown that the REDWC approach is easily applicable to multiplier arrays resulting in designs using approximately 45% less gates than a competing approach [CJA89].

Our second approach to concurrent error detection uses conservative logic. A conservative logic gate exhibits a one-to-one correspondence between input and output, so that the input of a conservative logic gate can be uniquely determined from its output. Our new approach derives its capabilities from the conservative attribute of the gate. A gate is said to be conservative if its output has as many 1's as are present at the input.

The significance of this approach is that the primitive element, the conservative logic gate, has the attribute of supporting concurrent error detection which is not the case for typical elements such as AND and OR gates. For this reason, it is very easy to implement any arbitrary function that contains error detection. We have designed and fabricated VLSI adders and multipliers using conservative logic [SJA89a]. Area efficiency is not yet acceptable; we are experimenting with its improvement.

Our last approach uses parity prediction. Parity is a technique in which the number of ones in the set of outputs of a network is forced to be either odd or even. Corruption of a single bit of the output causes the total number of ones to deviate from the expected parity. Parity techniques are common to memory design but have not been generally applied to arithmetic and logic circuits. We have developed a technique that allows parity to be preserved within *general combinational logic circuits*. The theory supporting the technique has been developed; example circuits have been designed [SJA89b]. While resulting circuits are efficient, the design algorithm is not. We are attempting various optimizations of the algorithm.

With the three above approaches we expect to develop a library of building blocks which can detect an error. How does one build a system that utilizes the knowledge? We will characterize the properties of the blocks, and determine methods for interconnecting them so as to attain overall system fault tolerance and testability. In addition, we want to use the uninterpreted modeling techniques described in Section F.1.2 to model fault tolerance at the conceptual design stage. That is, at the stage at which a circuit is modeled as tokens passing among functional elements, and computed values are not known, we want to be able to assess the fault tolerance features and their impact on the overall performance [RJA89]. Further we want to repeat such automatic analysis as elements are incrementally replaced by an interpreted model, that is one in which actual signal values can be computed.

The proposed infrastructure improvements will increase the number of design seats in the Center for Semicustom Integrated Systems as well as providing a new probe capability for testing for faults. Both will further this project.

References

[BeD88] M. E. Benitez and J. W. Davidson, A Portable Global Optimizer and Linker, *Proceedings of the SIGPLAN Notices '88 Symposium on Programming Language Design and Implementation*, Atlanta, GA, June 1988, 329-338.

[CJA89] J. B. Carr, B. W. Johnson and J. H. Aylor, An Efficient Method for Concurrent Error Detection in Parallel Multiplier Arrays, *IEEE Journal of Solid-State Circuits (submitted)*, 1989.

[Dav86] J. W. Davidson, A Retargetable Instruction Reorganizer, *Proceedings of the SIGPLAN Notices '86 Symposium on Compiler Construction*, Palo Alto, CA, June 1986, 234-241.

[Dri83] K. Driscoll, Multi-Microprocessor Flight Control System, *Proceedings of the Fifth Annual Digital Avionics Conference*, Seattle, Washington, November 1983.

[JAH88] B. W. Johnson, J. H. Aylor and H. H. Hana, Efficient Use of Time and Hardware Redundancy for Concurrent Error Detection in a 32-bit VLSI Adder, *IEEE Journal of Solid-State Circuits 23*,1 (February 1988), 208-215.

[Joh89] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[RJA89] R. Rao, B. W. Johnson and J. H. Aylor, Uninterpreted Modeling of Fault-tolerant Architectures, *IEEE Transactions on Computer Aided Design of VLSI (submitted)*, 1989.

[RAE81] D. Rennels, A. Avizienis and M. Ercegovac, *Fault-Tolerant Computer Study*, Jet Propulsion Laboratory, California Institute of Technology, Final Report, Contract No. NAS7-100, February 1981.

[SeL80] R. Sedmak and H. Liebergot, Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration, *IEEE Transactions on Computers C-29*,6 (June 1980), 492-500.

[SJA89a] G. Swaminathan, B. W. Johnson and J. H. Aylor, Approaches to Concurrent Error Detection Using Conservative Logic, *IEEE Transactions on Computers (submitted)*, 1989.

[SJA89b] G. Swaminathan, B. W. Johnson and J. H. Aylor, An Algorithm for Parity Prediction in Combinational Circuits, *IEEE Transactions on Computers (submitted)*, 1989.

[WJW75] W. Wulf, R. K. Johnsson, C. B. Weinstock, S. O. Hobbs and C. M. Geschke, *The Design of an Optimizing Compiler*, American Elsevier, New York, NY, 1975.

[Wul88] W. A. Wulf, The WM Computer Architecure, *SIGARCH News*, March 1988.

[WuH89] W. A. Wulf and C. Hitchcock, *The WM Family of Computer Architectures*, Computer Architecture Conference (submitted), 1989.

[Wul89] W. A. Wulf, The WM Computer Architecture Definition and Rationale, Report No. Tech. Rep.-89-22, University of Virginia Computer Science, Charlottesville, Va., February 1989.

F.3 Real-Time Software

The goals of the real-time software group are two-fold. First, we will advance the state-of-the-art in the technologies to support the construction of real-time operating systems, communication networks, and database systems. We consider such systems from the point of view of fault tolerance as well as performance. Second, the research necessitates development of tools and software components useful to other research groups; we will develop them with the goal of distribution. Our greatest infrastructure need is to have an excellent, tool-rich software development environment.

F.3.1 The StarLite Project (Cook, Son)

It has never been possible to develop real-time operating systems efficiently and conveniently on a host environment. The StarLite [Coo86, Coo87, SoK89] software development system is a substantive step in that direction. StarLite hosts the interactive, incremental development of software generally. More particularly, it supports development when the hardware is not available, not yet in existence, or when the target hardware is inhospitable for software development. The project currently involves two faculty, 8 graduate students, and several undergraduates, and is funded by ONR, ARO, and IBM.

The approach is 1) to build atop the Unix operating system for portability; 2) to specify a virtual machine, an interpreter, on which the host and target software (being developed) execute so that calls to the host, target and underlying Unix routines can be freely interspersed; and 3) to specify a low-level interface to simulate target hardware. The latter includes provision for execution of the target software on single processors, multiple processors with shared memory, and distributed systems. Interfaces exist for virtual clocks, disks, Ethernet connection and other devices. All software developed above this hardware interface layer remains invariant with respect to execution in either the host or target environment.

The base environment exists, as does an initial set of development tools including a symbolic, window-based, multi-thread debugger; a visualization package; a Modula-2 compiler that compiles 20,000 LPM on a Sun 3; and a software library of over 200 modules. The compiler and run-time libraries are implemented in C to enhance the portability. The system has been ported to the Sun 4 and NeXT architectures. To port to any UNIX system, the compiler and runtime library must be recompiled and the runtime interface to the underlying window system must be rewritten. To port target code to its target hardware, it need only be recompiled.

The system has been used by multiple instructors to teach graduate and undergraduate operating system and database courses. Students routinely write and integrate their modules for synchronization, memory management, and process management into our test system. StarLite is well-suited for implementation of alternative algorithms for comparative experimentation. Typically, it has not been possible to compare features of different operating systems directly. Often one-of-a-kind hardware is unavailable, or there is no environment in which only a selected portion of an operating system could be implemented for experimentation. StarLite now provides a base for such activity.

A Unix-compatible operating system shell, plus utilities, has been implemented (9000 lines) in the StarLite environment. In addition, a database system has been implemented using StarLite. It was selected by IBM to validate the priority ceiling algorithm developed at CMU because the environment was conducive to the comparative experimentation needed [SoC89].

The next domain of application for StarLite is embedded real-time operating systems (by Cook) and real-time database systems (by Son). The long-term objective is to validate the use of StarLite as a basis for real-time software development and to package it for distribution for use in classes and research.

The real-time operating systems research will span several projects. The first project explores the tension between layered software and its execution performance. Operating system components are designed in layers with standard but very carefully crafted interfaces. Each layer, composed of a set of modules, is designed to provide a coherent function. New target software may be built up from any layer, even if there are higher layers already in existence. Scrupulous observance of layering and interfaces costs procedure call performance. Application requirements, such as protection, can affect implementation in many layers. Even if an upper layer of software is not used, its requirements may have a residual effect on the code size and performance of lower layers. We will investigate how best to partition the layers and define the interfaces to maintain an open architecture, while still retaining the ability to compile code that yields good performance.

Second, the function provided by a module may have alternative implementations. We will explore the definition and management of multiple implementations with the intent of creating a systems generator that will automatically select implementations from a module library based on specified application requirements and a given target architecture. The context of this investigation will be that of real-time software which executes "very close" to its target hardware. Hardware changes require changes in related software. Our research plan calls for the creation of a library of alternative implementation modules in the domain of fine-grained synchronization and locking for real-time multiprocessors.

Third, we will extend the Unix implementation to see how real-time guarantees might be incorporated.

The real-time database research component of the StarLite project addresses real-time response guarantees while collecting, updating, and retrieving shared data. State-of-the-art database management systems are not used in real-time applications due to two inadequacies: poor performance and lack of predictability. Transaction scheduling does not consider response requirements, and data tables are locked indiscriminately to assure database consistency. Basically locks, as defined today, and time-driven scheduling are incompatible. Low priority transactions can and will block higher priority transactions, which lead to response requirement failures. We seek new techniques to manage database consistency while meeting real-time constraints [Son88, SoA89, SoC89].

Our approach is to create a model of real-time databases. We will then derive characteristics of transactions, data objects, their storage and accessing, and transaction schedulers. Based on this analysis of characteristics that affect responsiveness, we will develop database definitions as well as storage and processing techniques upon which we can define a transaction scheduler which can make and meet real-time guarantees of service, taking reliability requirements into consideration.

We will extend the database system built in the StarLite environment. It already provides concurrent transaction execution facilities including two-phase locking, timestamp ordering and priority-based contention protocols as the underlying synchronization mechanisms, as well as a multiversion data object control mechanism [SoK89]. We have validated the preliminary implementation. We will upgrade it while performing three closely related tasks:

- Upgrade the database library to enhance modularity so as to attain more component reusability.
- Develop a semantic model for real-time temporal knowledge of data objects and transactions in order to devise a new notion of "correct execution" of real-time transactions.
- Devise efficient schedulers and reliability control mechanisms that can be used as building blocks—having known specified performance and reliability characteristics—for implementing a real-time distributed database with predictable behavior.

The resulting reusable database component library housing components with known performance/reliability characteristics will be exportable to other database research sites.

F.3.2 Real-Time Communications (Weaver)

The Computer Networks Laboratory (CNL) at the University of Virginia was established in 1980 to pursue research and development in computer networks. We have a broad range of experience with the design, analysis, implementation, modeling, and performance measurement of computer networks and their protocols. The long-term goal of CNL is to solve the research and development problems that will enable medium-speed (10-100 Mbit/sec) and high-speed (Gbit/sec) networks to be placed into routine use in commercial and military real-time environments.

CNL currently consists of 10 graduate students under the direction of Professor Weaver; other faculty members participate on a project-by-project basis. To date the laboratory has completed 45 research projects, graduated 41 students with M.S. and Ph.D. degrees, and published over 60 journal and conference papers. Six current projects, their sponsors, and recent publications include:

- SeaNET: a real-time network for ships (Sperry Marine, Proteon) [SWC88, WeM89]
- AirNET: a real-time network for aircraft (NASA-Lewis) [CoW87]
- performance measurements of ISO protocols (Intel, Motorola) [StW88a, StW88b]
- SAFENET: a real-time network for military ships (Navy) [PeW87]
- Xpress Transfer Protocol (Protocol Engines, Navy, NASA, Sperry Marine) [WeS89a]
- prototype communications system for the NASA Space Station (NASA-JSC) [WeS89b]

Using LAN technology and protocols for real-time communications introduces some major research problems: (a) latency control; (b) user interface; (c) efficient transport protocols; (d) dynamic message importance; (e) rate control; (f) multicast transmission; (g) global sense of time; (h) distributed process synchronization; (i) protocol specification, verification, and compilation; and (j) rapid prototyping of protocols in hardware.

We will have new proposals for the first five topics within three years and will show progress on all ten topics in 5-8 years. Capitalizing upon our extensive implementation experience, our plan is to make routine the use of medium- and high-speed communications in both local (campus, ship, aircraft, factory) and long-haul (coast-to-coast, satellite-to-ground) real-time applications. Three efforts are described below.

LAN Protocols for Real-Time Control Systems: Distributed control systems are now commonly implemented using local area networks rather than point-to-point wiring or avionics busses. A common problem is that LANs generally do not provide the low latency typical of hardwired connections. We have attacked this problem in two ways: reducing the complexity of the user interface, and providing latency control as a feature of the underlying transport protocol.

User Interface. In a typical commercial network, passing a message from the user to the network interface hardware may involve multiple steps, each incurring the cost of multiple message copies or DMA transfers and multiple interrupts. We measured two commercial real-time systems whose end-to-end delay for 100-byte messages was 5 to 10 ms. In contrast, our SeaNET system—a real-time network for ships—collapses the entire user interface into twelve commands which connect the user directly to the network interface. Our documented end-to-end delay for a 100-byte message is 680 microseconds. While skillful implementation is a factor, performing high-speed communications requires identification of the essential services of real-time systems, a new design philosophy for user/system buffer pools which reduces copying, and a new strategy for reducing message fragmentation.

Latency Control. Traditional transport protocols (e.g. ISO TP4 and TCP) are optimized for throughput, not latency. Yet real-time systems are more affected by end-to-end delay than by raw transmission capacity. The research problem, then, is *latency control.* We are proposing and analyzing a new class of transport protocols, tentatively called "latency division multiplexing." Like time-division and frequency-division multiplexing, the concept is to divide the available bandwidth into latency classes. Performance guarantees are made to every message permitted to a class. Unlike most other protocols, service is pre-emptive. This trades off reduced throughput for improved latency control. Our results will be in the form of a new transport protocol for real-time systems, complete with analytic analysis and actual measurement of its performance characteristics.

Message Importance and Pre-emption in Real-Time Systems: For real-time systems it is generally important to assure that, at every moment, the system as a whole is working on its most important task(s). Conventional real-time systems lack effective mechanisms to recall previously made scheduling decisions. For example, most network access protocols support priority, but only at the granularity of message frames. Once a message has begun segmentation and transmission, it is generally impossible to pre-empt it in favor of a more recent but more important message. At the transport layer, TCP and TP4 provide only two levels of importance, and they are not pre-emptive. We are investigating two approaches: changing an existing protocol, and creating a new protocol with the desired capability.

Xpress Transfer Protocol (XTP). XTP is a transport protocol designed from the outset for implementation in silicon as the "protocol engine." As originally proposed, XTP did not include specific support services for real-time systems; our contribution to date has been an analysis of the needs of such systems and a proposal for essential real-time services. Our proposal was largely adopted by XTP's designers in the July 1989 version [Pro89], which added static and dynamic message priorities to the protocol definition. We are investigating priority assignment algorithms that dynamically assign message importance based on a) multiple, competing goals which change with time, and b) message age.

Our Real-Time Protocol. Armed with our experiences in developing several "real" real-time networks, we are specifying a new communications protocol that includes all the essential services we have identified, and yet is simple enough to be implemented in VLSI. This protocol would be limited to interconnected LANs (e.g., token rings) and limited topologically to the domain of ships, aircraft, space vehicles, and small factories. Our near term goal is to produce our new protocol's specification, implement and refine it in software, and compare its performance *in the domain of real-time systems* to that of TP4, TCP, and XTP.

Protocol Compilation and Verification: There is no such thing as the "perfect" protocol; each one represents a particular combination of features and a compromise of performance attributes.

Our work in real-time systems suggests that there will always be a need for communications protocols with differing combinations of features and performance. We envision a future in which one can design new protocols quickly, describe them in a new protocol specification language, and compile the description into running code. The resulting code could either be executed conventionally on a host or front-end processor, or it could be reduced to a custom VLSI circuit. Our current research task is to develop the necessary theory for protocol specification, verification, validation, and compilation.

In the context of these projects, we expect to experiment with circuit design. As message communication speeds increase, protocol execution speed must also increase. To perform research in the medium to high-speed arena, we require the System Integration Laboratory in order to combine off-the-shelf and custom components for experimentation. To date, CNL has been restricted to the use of commercial components with no capability of adapting them.

F.3.3 Software Fault Tolerance for Real-Time Systems

Certain applications of computers are referred to as *crucial* because failure of the computer system could endanger human life, damage expensive equipment, or otherwise lead to extensive financial loss. Such applications are usually control systems, and are embedded and operate in real time. Although sophisticated techniques for building fault-tolerant hardware systems have produced computer architectures of great reliability [Hop78, SiS82, Wen78], no corresponding techniques exist for building software with assured, adequate reliability. Software faults are *design* faults and differ fundamentally from faults due to hardware degradation. Therefore, the techniques used to achieve hardware reliability do not apply directly to software [AmK89].

As part of a long term research project in software reliability we are investigating software fault tolerance. We have shown that one technique, *N*-version programming [Avi85], has several difficulties that make its use in crucial applications very unwise [AKB89, AmK90, BKL89, BKL90, KnL86a] although some reliability gain has been demonstrated [KnL86b]. Almost all proposed techniques for building fault-tolerant software attempt to survive essentially *any* fault. This contrasts sharply with other engineering disciplines where the tolerance achieved is with respect to specific fault classes. The attempt to cope with all possible faults is the basis of much of the difficulty in implementing the methods and of the unpredictability in their performance. A notable exception is the work on robust data structures [TMB80] in which provision is made to cope only with faults that result in data-structure defects. These techniques are provable and so their performance is guaranteed.

During the next several years we plan to continue research into the general area of software fault tolerance for specific fault classes. Two projects are described here.

First, we will develop techniques for tolerating specific classes of faults. We intend to develop a classification scheme of important types of software faults in real-time systems such as failure to meet real-time deadlines, deadlock, and livelock. For each, we will develop software structures that can be shown to cope with the associated fault type. These techniques will lead to systems that have predictable performance for (only) specific fault categories and an assured performance improvement over non-fault-tolerant systems.

Most existing techniques for tolerating software faults rely upon redundant software. Functionality is replicated with multiple, independently developed implementations. Such techniques are said to incorporate *design diversity* because the different implementations are assumed to incorporate different designs. In contrast, our second project involves a technique for tolerating software faults that relies on *data* diversity [KnA88]. A computer program is frequently a many-to-one mapping, and when a fault is encountered, correct operation might be possible if the input data is replaced by different data that should produce equivalent output. The fault is tolerated if the new data causes the program to execute a different path that does not contain the original fault or any other faults. Data diversity has been investigated sufficiently to show feasibility for a number of applications. Its performance has been shown to be promising in a pilot study. We will develop the technique by extending the range of application, by refining the performance model of the method, and by experiments to ascertain the realized reliability improvement.

The computing infrastructure for the fault tolerance analysis and experimentation, as well as for the other real-time projects described, is resource intensive. We require general purpose computation, storage, and display facilities. We require an excellent software development environment and analytic tools.

References

[AKB89] P. E. Ammann, J. C. Knight and S. S. Brilliant, The Effect Of Comparison Checking On Multi-Version Software Performance, *IEEE Transactions on Software Engineering*, in review 1989.

[AmK89] P. E. Ammann and J. C. Knight, Issues Influencing The Use Of N-Version Programming, *Proceedings of IFIP '89: Eleventh World Computer Conference*, San Francisco, CA, August 1989.

[AmK90] P. E. Ammann and J. C. Knight, Design Fault Tolerance, *Journal of Reliability Engineering and System Safety*, to appear 1990.

[Avi85] A. Avizienis, The *N*-Version Approach to Fault-Tolerant Software, *IEEE Transactions on Software Engineering SE-11*,12 (December 1985).

[BKL89] S. S. Brilliant, J. C. Knight and N. G. Leveson, The Consistent Comparison Problem In *N*-Version Software, *IEEE Transactions on Software Engineering SE-15*,11 (November 1989).

[BKL90] S. S. Brilliant, J. C. Knight and N. G. Leveson, Analysis of Faults in an *N*-Version Software Experiment, *IEEE Transactions on Software Engineering*, to appear 1990.

[CoW87] M. A. Colvin and A. C. Weaver, A Real-Time Messaging System for Token Ring Networks, *SOFTWARE - Practice and Experience* , December 1987, 885-897.

[Coo86] R. P. Cook, StarLite, A Visual Simulation Package for Software Prototyping, *Second Symposium on Practical Software Environments*, Dec 1986, 102-110.

[Coo87] R. P. Cook, StarLite, A Network-Software Prototyping Environment, *Symposium on the Simulation of Computer Networks*, Aug 1987, 178-184.

[Hop78] A. L. Hopkins et al, FTMP - A Highly Reliable Fault-Tolerant Multiprocessor For Aircraft, *Proceedings of the IEEE 66*(October 1978), 1221-1239.

[KnL86a] J. C. Knight and N. G. Leveson, An Experimental Evaluation of the Assumption of Independence in Multi-version Programming, *IEEE Transactions on Software Engineering SE-12*,1 (January 1986).

[KnL86b] J. C. Knight and N. G. Leveson, An Empirical Study of Failure Probabilities in Multi-Version Software, *Digest FTCS-16: Proc. 16th Int. Symposium on Fault Tolerant Computing*, Vienna, Austria, July, 1986, 165-170.

[KnA88] J. C. Knight and P. E. Ammann, Data Diversity: An Approach To Software Fault Tolerance, *IEEE Transactions on Computers C-37*,4 (April 1988).

[PeW87] J. H. Peden and A. C. Weaver, Performance of Priorities on an 802.5 Token Ring, *IACM SIGCOMM*, Stoweflake, Vermont, August 1987.

[Pro89] Protocol Engines Inc., XTP Protocol Definition, Release 3.4, July 1989.

[SiS82] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.

[SWC88] R. Simoncic, A. C. Weaver, B. G. Cain and M. A. Colvin, SEANET: A Real-Time Communications Network for Ships, *International Conference on Mini and Microcomputers*, Miami Beach, FL, December 14-16, 1988.

[Son88] S. H. Son, Real-Time Database Systems: Issues and Approaches, *ACM SIGMOD Record, Special Issue on Real-Time Database Systems 17* ,1 (March 1988).

[SoC89] S. H. Son and C. Chang, Distributed Real-Time Database systems: Prototyping and Performance Evaluation, *International Symposium on Database Systems for Advanced Applications*, April 1989, 251-258.

[SoK89] S. H. Son and Y. Kim, A Software Prototyping Environment and Its Use in Developing a Multiversion Distributed Database System, *18th International Conference on Parallel Processing*, St. Charles, Illinois, August 1989.

[SoA89] S. H. Son and A. Agrawala, Distributed Checkpointing for Globally Consistent States of Databases, *IEEE Transactions on Software Engineering (to appear) 15*,10 (October 1989).

[SoC89] S. H. Son and R. P. Cook, Scheduling and Consistency in Real-Time Database Systems, *Sixth IEEE Workshop on Real-Time Operating Systems and Software*, Pittsburgh, Pennsylvania, May 1989, 42-45.

[StW88a] W. T. Strayer and A. C. Weaver, Performance Measurement of Data Transfer Services in MAP, *IEEE Networks Magazine*, May 1988.

[StW88b] W. T. Strayer and A. C. Weaver, Performance Measurements of Motorola's Implementation of MAP, *13th Conference on Local Computer Networks*, Minneapolis, MN, October 10-12, 1988.

[TMB80] D. J. Taylor, D. E. Morgan and J. P. Black, Redundancy in Data Structures: Improving Software Fault Tolerance, *IEEE Transactions on Software Engineering SE-6*,6 (November 1980).

[WeM89] A. C. Weaver and J. F. McNabb, A Real-Time Monitor for Token Ring Networks, *MILCOM'89*, Boston, MA, October 15-18, 1989.

[WeS89a] A. C. Weaver and R. Simoncic, Implementing XTP for the NASA Space Station, *IFIP Workshop on High Speed Protocols*, Zurich, Switzerland, May 1989.

[WeS89b] A. C. Weaver and R. Simoncic, Communications for the NASA Space Station, *14th Local Computer Networks Conference*, Minneapolis, MN, October 10-12, 1989.

[Wen78] J. H. Wensley et al., SIFT, The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control, *Proceedings of the IEEE 66*(October 1978), 1240-1254.

F.4 Parallel Systems

Hardware for parallel systems is relatively straightforward to construct, as evidenced by the many commercial parallel systems currently on the market. In contrast the software systems for these machines remain relatively unsophisticated. In the Institute for Parallel Computation (IPC), we have three medium scale commercial parallel systems: a 128 node NCUBE/ten hypercube, a 32 node Intel iPSC/2 hypercube, and a 32 node BBN GP-1000 "butterfly". Our overall goal is to develop programming environments and software sub-systems needed to use these machines *effectively*. Research activities include basic work in parallel database systems, parallel discrete event simulation, parallel programming languages and environments, and parallel high-performance input/output subsystems, and parallel genetic algorithms. In addition the IPC hosts more application oriented research in parallel data correlation and fusion algorithms, parallel terrain mapping algorithms, parallel vision systems, and neural net algorithms. In this section we describe three of these research projects.

F.4.1 ADAMS, A Parallel Database Interface (Pfaltz, Son, French, Grimshaw)

The goal of the ADAMS—the Advanced Data Management System—project is to create a standard interface which will support the parallel access and management of persistent data. We envision ADAMS becoming a standard interface between programs written in algorithmic languages, such as C, Pascal, Ada, and Fortran, to persistent data. In commercially available information systems, the data operators can be predicted and implemented within the database. In scientific computations, the scientist must program the data operators. Hence, the scientists' programs would benefit from a gracefully integrated data access capability. Such an interface obviates the need for database languages apart from the algorithmic language. Persistent Algol [BuA86] and ODE [AgG89], which have been proposed to handle persistent data in existing languages, have a similar goal. The significant differences are that ADAMS is designed to (a) interface with all commonly used scientific languages, (b) share data between distinct users, and (c) provide a parallel interface to the data. It is our objective to develop a complete programming environment to support coarse grained parallelism. The ADAMS project is focused on five key sub-problems:

Data description and access. One needs a data description and access syntax that is both flexible and simple. The ADAMS language, described in [PSF88], consists of only five basic constructs *class, set, attribute, map,* and *codomain.* Every ADAMS *element* is an instance of, or belongs to, a *class*. *Classes* can be hierarchically declared within an executing program, with subclasses inheriting the properties of their superclasses. *Set, attribute,* and *map* are the remaining basic ADAMS classes. A *codomain* describes a set of data values.

A prototype version was first implemented so that test applications could be written and analyzed. This led to several important design modifications. An object-oriented C++ version is currently being implemented with expected completion by October.

Naming. An item or collection of data is referenced by *name*. When data is persistent, names must also be persistent. We have developed a dictionary concept [PFW88] to implement a large persistent hierarchical name space. Data name scope can be specified to be a block, a task, global, or system-wide. Automatically managing such a hierarchical name space so that naming conflicts are avoided is a central research goal.

ADAMS is intended for scientific applications needing access to many data values having the same form. To expand the ADAMS name space we use generally subscripted names. The subscript itself does not bind the storage location of a value. Two values whose names differ only by subscript may not be in predictable, relative storage locations. We are experimenting with innovative techniques for associating groups of subscripted, named values to storage locations.

Parallel data storage. To achieve a high degree of parallel access to data by processes executing in parallel, data must reside on multiple devices. Our approach is to balance load on devices dynamically via data migration in response to patterns of data usage. With such a strategy the logical record structure no longer tightly binds the elements of that record to be co-located on a single device. ADAMS avoids record/file structures altogether. Data elements migrate within the storage system in response to patterns of data usage. The benefits of such data distribution, for example with access to set operations, was examined in [PFS89]. Issues of parallel data distribution are discussed in [Fre89, SCR89].

Rapid access. If data items can float in the storage environment, then new methods of access must be developed to find and access them. Older indexed access schemes become impractical. We have developed Compact 0-complete trees (or O-trees) for this purpose [OrP88]. O-trees are a kind of B-tree, in which the space consuming index keys have been replaced by a single 8 bit (byte) surrogate. These trees have been described by Ed McCreight as the "most significant improvement of B-trees in the last 10 years". They are the basis for our implementation of data element access, set representation, and identifier subscripting [OrP89].

Integrity and reliability. If multiple processes can create, access, and update data—the integrity of data must be assured. Also, if the database is distributed, as we anticipate for efficiency purposes, it must be fault-tolerant. Reliability achieved through efficient checkpointing [Son88b, SoP88], and data replication [Son89] have been explored. We are testing various reliability concepts in the StarLite prototyping environment [Son88a, SoK89].

Parallel environment. Large grain expressions in ADAMS' statements must execute in parallel to achieve our high-performance goals. To explore this aspect we have selected the Mentat prototype system, designed to support large-grain parallel computation [GLT87, GrL87, GrL88]. It consists of the Mentat Programming Language (MPL), an object-oriented programming language based on C++, and the Mentat run-time system. Programmers are responsible for providing information for object classes amenable to parallel processing, and the compiler and run-time system are responsible for managing parallelism with its associated communication and synchronization. Mentat provides a cost-effective compromise between fully explicit and automatic techniques. The Mentat compiler and run-time systems are suitably instrumented for performance monitoring. Mentat executes on a network of Sun workstations as well as the Intel hypercube. We are currently working on a production version of the system that will include both an instrumented run-time system as well as an MPL compiler.

F.4.2 Parallel Simulation (Reynolds)

Parallel simulation is the parallel execution of discrete event simulations. Beginning with the models of Chandy and Misra [ChM81] and Peacock, et al. [PWM79], a number of approaches have been described for coordinating cooperating processes so that the outcome of a parallel simulation is determinate. There are two problems with earlier work. There have been a number of characterizations and classifications of parallel simulation protocols, e.g. "optimistic" vs. "conservative", and of the applications, e.g. "queueing" vs. "non-queueing". But there has not been a comprehensive classification. Second, the effectiveness of parallel simulation protocols is usually determined by comparing individual experiments conducted in isolated environments. Such one-shot experiments do not advance general understanding.

We have found a way to characterize both protocols and applications in a comprehensive and useful way [Rey88, RWF89]. This has led us to design a parallel simulation testbed, SPECTRUM, for comparing protocols in a realistic context of applications. SPECTRUM is implemented on both a 32-node BBN GP-1000 parallel computer and an 32-node Intel iPSC/2 hypercube [ReD89]. It enables the rapid implementation of protocols in a common, instrumented environment hosting a range of applications. It was our early experience with the testbed which made it evident that a set of design variables for applications had to be explored; it is inadequate to consider protocols in a vacuum. SPECTRUM is a vehicle for our future research which generally aims to improve the performance and effectiveness of parallel simulation techniques. Three of our current research projects are described below.

Experimental and analytic comparison of protocols. SPECTRUM currently hosts several protocols: SRADS, Chandy-Misra null messages, and a new protocol called SRADS with local rollback. Implemented applications include queueing networks, logic networks, and a battlefield simulation. We continue to augment both the library of protocols and of applications as well as the tools SPECTRUM provides for analysis. We will continue to compare and contrast protocols with the goal of understanding the interplay of classes of protocols and classes of applications. By considering additional design variables, such as adaptability, risk, accuracy, and knowledge embedding, dissemination, and acquisition, we can identify as yet unstudied points in the protocol design space. We intend to polish the user interface to SPECTRUM so that other UVa researchers will use it as a simulation tool. We need the diverse applications this use will bring. One early application we seek is the integration of a fault simulator that is in current use in the Center for Semicustom Integrated Systems. We are also pursuing complementary analytic work. Collaboration with D. Nicol of the College of William and Mary has resulted in the development of new analytic techniques for analyzing protocols with varying amounts of aggressiveness.

Development of new protocols. We have developed protocols in the past: SRADS [Rey82] and SRADS with local rollback. Recently, we have been working with K. M. Chandy at Caltech on a new approach developed by Chandy called "space-time".

Operating system and hardware support for parallel simulation. Large discrete-event simulations are computation and communication intensive, even on parallel machines. They can benefit from specially designed operators for very frequently repeated operations and for easing physical resource contention. We have defined efficient access operators for shared data structures such as linked lists and vectors. More recently, we designed a new interconnection network and memory module that supports parallel operations, *parops*, on shared data structures without blocking. Parops are composite memory instructions consisting of one or more operations on shared variables. Multiple parops can be executed by competing processes in a non-interfering manner. Yet from the perspective of each process, its own parop is executed as an indivisible operation. Our work involves operators similar to those in the Yale Fluent machine [Ran87]. In the future we will build interconnection modules and smart memories using the integrated circuit design facility and Systems Integration Laboratory.

F.4.3 Parallel Genetic Algorithms (Cohoon, Martin, Richards)

Combinatorial optimization problems preclude the direct application of many standard optimization techniques, e.g., gradient-based hill climbing. Their combinatorial nature has caused some researchers to turn to probabilistic search mechanisms, such as simulated annealing [KGV83] and genetic algorithms [Gol89]. We are studying a novel formulation of genetic algorithms that is motivated by the concept of punctuated equilibria in evolution theory [ElG72] and the desire to use effectively large scale, distributed-memory, message-passing, parallel-processing systems. We call this new approach, *GAPE.*

The *genetic algorithm* (GA) paradigm [Hol75] has been proposed to generate solutions to a wide range of optimization problems, including control systems, function optimization, and combinatorial problems [CoP87, CHM87, Gol89]. In a genetic algorithm, a *population* of solutions to the problem at hand is maintained and allowed to *evolve* through successive *generations.* A suitable encoding of each solution allows computation of the *fitness,* i.e., a measure of the solution's *competence,* and manipulation to form new solutions. Manipulation either merges two solutions from the current generation via a *crossover* operator or alters an individual solution using a *mutation* operator. Solutions to be included in the next generation are probabilistically selected according to their fitness from both the current generation and the new solutions. These capabilities provide the means to create a sequence of generations.

There are many simple ways to implement a sequential genetic algorithm on a global shared-memory multiprocessor, e.g., selecting and crossing-over pairs of solutions in parallel, and mutating solutions in parallel. Such implementations result in a linear speedup at best, and will perform unacceptably poorly on local-memory, message-passing, distributed systems. In order to investigate a more effective paradigm, in addition to one that is immediately suitable for mapping genetic algorithms onto a distributed processor, we have turned our attention to the theory of punctuated equilibria.

The theory of *punctuated equilibria* [ElG72] has been proposed to resolve certain paleontological dilemmas in the geological record, in particular, the rapid evolution of new species. Punctuated equilibria stresses that a powerful method for generating new species is to thrust an old species, i.e., one from a stable environment into a new environment, where change is beneficial and rewarded.

Our approach uses major iterations called, *epochs.* During an epoch each processor, disjointly and in parallel, executes the genetic algorithm on its subpopulation. After each epoch there is a phase during which each processor copies randomly selected subsets of its population to neighboring processors, then probabilistically selects a set of solutions that survive to be its initial subpopulation at the beginning of the next epoch.

The relationship to punctuated equilibria is the following. Each processor corresponds to a disjoint *environment* as characterized by the mix of solutions in it. Different environments arise if the fitness measure is defined relative to the current local population. In this way, exchanging sets of solutions between local populations will alter the evaluation of the members of the local populations, and introduce new competitors thereby effecting the desired rapid evolution of new species. After some generations we expect to see the emergence of some very fit species. Then a catastrophe occurs and the environments change. This is simulated by having representatives of geographically adjacent environments regroup to form the new environments. By varying the amount of redistribution, we can control the amount of disruption.

We have developed a distributed genetic algorithm based on punctuated equilibria, experimented with the NP-complete Optimal Linear Arrangement problem [CHM87] and the floor-plan design problem [CHM88], and empirically demonstrated that our paradigm is much more effective than straightforward parallel versions of a sequential genetic algorithm. Further, we have implemented the system on a hypercube and have developed appropriate representations for additional problem domains. We are now ready to consider several research directions. Two issues will be addressed immediately. First, *GAPE* depends fundamentally on having distinct environments providing speciation pressure on interacting subpopulations. We will investigate methods to create and modify dynamically the needed environments. Second, the concept of convergence is basic to the definition of the successive epochs. We will investigate the meaning of convergence in this domain of probabilistic search in which the progression is driven, at least partially, by random selection operators.

References

[AgG89] R. Agrawal and N. H. Gehani, ODE (Object Database and Environment): The Language and the Data Model, *Proceedings 1989 ACM SIGMOD Conference on the Management of Data*, Portland, OR, June 1989, 36-45.

[BuA86] P. Buneman and M. Atkinson, Inheritance and Persistence in Database Programming Languages, *Proceedings 1986 ACM SIGMOD Conference on the Mangement of Data*, Washington, DC, May 1986, 4-15.

[ChM81] K. M. Chandy and J. Misra, Asynchronous Distributed Simulation via a Sequence of Parallel Computations, *Comm. ACM 24*,4 (Apr. 1981), 198-205.

[CoP87] J. P. Cohoon and W. D. Paris, Genetic placement, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems CAD-6*,6 (November 1987), 956-964.

[CHM87] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. Richards, Punctuated equilibria: A parallel genetic algorithm, *Second International Conference on Genetic Algorithms and Their Applications*, Cambridge, MA, 1987, 148-154.

[CHM88] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. Richards, Floorplan design using distributed genetic algorithms, *IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, 1988.

[ElG72] N. Eldredge and S. J. Gould, Punctuated equilibria: An alternative to phyletic gradualism, in *Models of Paleobiology*, Freeman, Cooper and Co., 1972, 82-115.

[Fre89] J. C. French, A Global Time Reference for Hypercube Multicomputers, *Proceedings 4th Conference on Hypercube Concurrent Computers and Applications*, Monterey, CA, Mar. 1989.

[Gol89] D. E. Goldberg, *Genetic Algorithms*, Addison-Wesley, 1989.

[GLT87] A. S. Grimshaw, J. W. S. Liu and M. Thomas, Mentat: A Prototype Macro Data-Flow System, *Proceedings of the 2nd Workshop on Large Grain Parallelism*, Oct. 1987.

[GrL87] A. S. Grimshaw and J. W. S. Liu, Mentat: An Object-Oriented Data-Flow System, *Proceedings of the 1987 Object-Oriented Programming Systems, Languages and Applications Conference*, Oct. 1987, 35-47.

[GrL88] A. S. Grimshaw and J. W. S. Liu, The Mentat Programming Language and Architecture, *Proceedings Workshop on Future Trend of Distributed Computing Systems*, Hong Kong, Sept. 1988.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[KGV83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science 220*,4598 (May 13, 1983), 671-680.

[OrP88] R. Orlandic and J. L. Pfaltz, Compact 0-Complete Trees, *Proceedings of the 14th Very Large Databases Conference*, Long Beach, CA, Aug. 1988, 372-381.

[OrP89] R. Orlandic and J. L. Pfaltz, Analysis of Compact 0-Complete Trees: A New Access Method to Large Databases, *Proceedings 7th FCT Conference*, Szeged, Hungary, Aug. 1989.

[PWM79] J. K. Peacock, J. W. Wong and E. Manning, *Computer Networks*, North Holland Publications, 1979.

[PFW88] J. L. Pfaltz, J. C. French and J. L. Whitlatch, Scoping Persistent Name Spaces in ADAMS, IPC TR-88-003, Institute for Parallel Computation, University of Virginia, June 1988.

[PSF88] J. L. Pfaltz, S. H. Son and J. C. French, The ADAMS Interface Language, *Proceedings 3rd Conference on Hypercube Concurrent Computers and Applications*, Pasadena, CA, Jan. 1988, 1382-1389.

[PFS89] J. L. Pfaltz, J. C. French and S. H. Son, Parallel Set Operators, *Proceedings 4th Conference on Hypercube Concurrent Computers and Applications*, Monterey, CA, Mar. 1989.

[Ran87] A. G. Ranade, How to Emulate Shared Memory, *IEEE Symposium on Foundations of Computer Science*, Los Angeles, 1987, 185-194.

[Rey82] P. F. Reynolds, A Shared Resource Algorithm for Distributed Simulation, *Proceedings of the Ninth Annual International Computer Architecture Conference*, Austin, Texas, April, 1982, 259-266.

[Rey88] P. F. Reynolds, A Spectrum of Options for Parallel Simulation Protocols, *Proceedings of ACM Winter Simulation Conference*, December 1988.

[RWF89] P. F. Reynolds, C. F. Weight and J. R. Fidler, Comparative Analyses of Parallel Simulation Protocols, *Proceedings of the ACM Winter Simulation Conference*, to appear December 1989.

[ReD89] P. Reynolds and P. Dickens, SPECTRUM: A Testbed for Parallel Simulation Algorithms, *Proceedings 4th Conference on Hypercube Concurrent Computers and Applications*, Monterey, CA, Mar. 1989.

[Son88a] S. H. Son, A Message-Based Approach to Distributed Database Prototyping, *Fifth IEEE Workshop on Real-Time Software and Operating Systems*, Washington DC, May 1988, 71-74.

[Son88b] S. H. Son, Efficient Decentralized Checkpointing in Distributed Database Systems, *21st Hawaii International Conference on System Sciences* 2(Jan. 1988), 554-560.

[SoP88] S. H. Son and J. L. Pfaltz, Reliability Mechanisms in ADAMS, *Proceedings 3rd Conference on Hypercube Concurrent Computers and Applications*, Pasadena, CA, Jan. 1988.

[Son89] S. H. Son, A Resilient Replication Method in Distributed Database Systems, *Proceedings IEEE INFOCOM '89*, Ottawa, Canada, Apr. 1989, 363-372.

[SoK89] S. H. Son and Y. Kim, A Software Prototyping Environment and its Use in Developing a Multiversion Distributed Database System, *International Conference on Parallel Processing*, St. Charles, IL, Aug. 1989.

[SCR89] S. H. Son, R. Cook and J. Ratner, Communication Paradigms for Message-Based Multicomputer Systems, *Proceedings 4th Conference on Hypercube Concurrent Computers and Applications*, Monterey, CA, Mar. 1989.

F.5 Application Systems

Two applications are discussed: computer vision and the development of a computational geometry toolbox. Both require an excellent software development environment.

F.5.1 Computer Vision in Dynamic Environments (Martin, Olson)

Developing vision systems that allow machines to interact intelligently with complex and changing environments is a problem of fundamental importance. The changing nature of the environment demands the analysis of dynamic visual data, i.e., time-ordered sequences of images such as the frames of a video signal. Our work is in the area of *dynamic scene analysis* [ADM81, MaA78, MaA88]. Further, we emphasize *focus of attention* mechanisms in vision systems based on parallel processing [TaM86, TaM89].

In the areas of machine perception and artificial intelligence, many systems are based on the concept of parallel processes cooperating in a quasi-independent manner. The term ''quasi-independent'' means that individual processes operate independently, but the final result is dependent on the actual order of operation of the processes. At one level controlling the attention of the system is just a matter of efficient resource allocation. At a more fundamental level are problem solving techniques which advance the system toward a solution.

The processing component of our vision system is a set of cooperating *agents* [TaM86, TaM89]. Each agent is a quasi-independent process capable of guiding the application of its processing resources to the primary objects of interest in the environment. The guidance constitutes the selection of a subimage within each image in the sequence. Each agent restricts its processing to selected subimages. The primary data structure used for guidance is a multiresolution image pyramid. It is a sequence of images of different granularity of resolution. At the base is the original image with fine grain resolution; other images have coarser resolution of ''averaged'' versions of the original image. An agent analyzes subimages of coarse resolution to guide the application of its processing resources in the finer resolution images of the pyramid. This control function is taken as an analog to the human vision system in which peripheral, i.e., coarse resolution, vision often serves to direct foveal, i.e., fine resolution, vision. Of course, the underlying data is an image sequence, so the pyramid structure evolves with time. *Cooperating* agents communicate via a blackboard.

We have two research objectives. First, we will develop effective operations on multiresolution imagery in the time-varying context. Secondly, in the long range this research will lead to an understanding of the fundamental nature of closely coupled cooperation of agents for the interpretation of visual information from dynamic environments. Our research program will involve specific applications. For each, we will determine the major dynamic characteristics of interest in the application and the resource demands of detecting, tracking and understanding those characteristics. Then, we will design multi-agent solutions that solve the problem while operating within the computational constraints. Our first application is tracking red blood cells in order to determine the location of capillaries. Experiments are concerned with various blood flow measures and the change in those measures under imposed conditions, for example the presence of particular chemicals. The development of computational structures capable of monitoring those measures in ''real-time'' will be a major advancement in that experimental methodology.

F.5.2 The Computational Geometry Toolbox (Cohoon, Martin, Richards, Salowe)

Computational geometry is concerned with manipulating, processing, and examining geometric objects [PrS85]. Most research has produced isolated algorithms, but there is no cohesive system to host these algorithms in an integrated way. For example, suppose a software researcher is interested in creating an effective software system for dealing with points in the plane where required operations include: dynamically finding the convex hull, locating nearest-neighbor points, and finding Steiner minimal trees. It is not known whether an integrated system could be practically built using the best solutions for the various subproblems. Besides using incompatible data structures, the best algorithms often have running times whose large constants are obscured in asymptotic analysis. Further, the algorithms are typically sensitive to minor changes in input. We propose to investigate the fundamental issues involved in implementing an integrated library of practical algorithm implementations.

Our toolbox development will emphasize:

Dynamic Operations: Algorithms should allow for dynamic insertions and deletions.

Robustness: Algorithms will be formulated to be insensitive to minor perturbations in the input.

Simplicity: Algorithmic design will be economical in that control structures and data structures will be the simplest available.

Commonality: Data structures which are common or can be easily transformed across the algorithms will be used.

Our initial design will focus on simplicity, robustness and dynamics; the more challenging aspect of commonality will be dealt with later. A small core of computational geometry tasks will be considered: convex hulls, Voronoi diagrams, Steiner trees and VLSI routing. We will analyze existing algorithms found in the literature: convex hulls [OvL81, Sei86], Voronoi diagrams [EdS85, For86, GKL83, GuS85], Steiner trees [Ber87, CRS90, RiS89], and VLSI routing [CoH88, CoR88]. We expect to modify some algorithms extensively for implementation. Modularity will be emphasized in the design, so that algorithms can be added, deleted, and modified easily. Once the initial toolbox is complete, it will be expanded and changed based on feedback from its user community.

We will design the toolbox for three types of users: researchers in computational geometry, in vision, and in VLSI. The core of the toolbox is a library of geometric algorithms; these algorithms will be accessed either directly or through a graphical interface. For computational geometry researchers, the toolbox will serve as a platform for implementing new algorithms, a means for studying the interrelationships between computational geometry algorithms, and a visualization tool. VLSI and vision researchers are expected to access the library routines directly. The applications of computational geometry to VLSI and vision are described below.

Computational Geometry in VLSI Design: One of the principal phases of the VLSI design process is circuit layout. In this phase there are several steps to which computational geometry theory and tools can contribute greatly: partitioning, floor-planning and placement, and routing. Besides obvious application to VLSI routing, which is discussed earlier in this proposal, we have also successfully incorporated computational geometry techniques in partitioning and placement. For example, in one ongoing placement investigation we have found computational geometry algorithms and transformations important in determining both solutions for and estimates of interconnection requirements measured in wire density, total wire length, and longest wire length [CoP87]. Although minimal spanning trees algorithms are used typically in determining the estimators, more accurate estimators can be determined by constructing minimal Steiner trees for the various interconnections. Because the general Steiner tree problem is *NP*-hard, research has concentrated on exact or approximation algorithms for some restricted classes of rectilinear problem instances. We have developed recently both a fast approximation algorithm that uses many geometric constructions, and is effective for a broad class of rectilinear problem instances [Ric89], as well as a fast exact algorithm for terminals on the perimeter of a VLSI switchbox [CRS90].

Computational Geometry in Computer Vision: A simple definition of vision has been given as: ''to know what is where by looking'' [Mar82]. It is geometry that describes the fundamental relationships to be determined by vision processes. Computer vision derives and manipulates geometric information characterizing the external environment. For a system to know ''where'' things are, it must be able to operate effectively and efficiently upon the geometric relationships imposed by the spatial configuration of the sensed environment.

For a system to know ''what'' things are in that environment constitutes *recognition.* Though less obvious, recognition also requires computational geometry. The geometry is determined by a ''parameter space'' derived from transformations on sensor measurements. For example, multispectral sensors aboard satellites result in multidimensional ''signatures'' for areas sampled from the surface of the earth. Recognition of ground use categories can then be formulated as a computational geometry problem, e.g., determine the convex hull of a set of points, in the multidimensional ''signature'' space.

The advantage of having a computational geometry toolbox in vision is the ready access to established procedures for the common computational geometry tasks that computer vision systems repeatedly demand. *Robustness* of algorithms used in computer vision is important because ''noise'' is an ever present component of the image sensing process. The *commonality* property of the toolbox will simplify the task of integrating toolbox algorithms into a vision system.

References

[ADM81] J. K. Aggarwal, L. S. Davis and W. N. Martin, Correspondence Processes in Dynamic Scene Analysis, *Proceedings of the IEEE 69*(1981).

[Ber87] M. W. Bern, Network Design Problems: Steiner Trees and Spanning k-Trees, Ph.D. Dissertation, Computer Science Division, Univ. of California at Berkeley, 1987.

[CoP87] J. P. Cohoon and W. D. Paris, Genetic Placement, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems CAD-6*,6 (November 1987), 956-964.

[CoH88] J. P. Cohoon and P. L. Heck, Beaver: A Computational-Geometry-Based Tool for Switchbox Routing, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems CAD-7*,6 (June 1988), 684-697.

[CoR88] J. P. Cohoon and D. S. Richards, Optimal Two-Terminal α-β Routing, *Integration: The VLSI Journal*, February 1988.

[CRS90] J. P. Cohoon, D. S. Richards and J. S. Salowe, An Optimal Steiner Tree Algorithm for a Net Whose Terminals Lie on the Perimeter of a Rectangle, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, to appear in 1990.

[EdS85] H. Edelsbrunner and R. Seidel, Voronoi Diagrams and Arrangements, *First Annual ACM Symposium on Computational Geometry*, 1985, 251-262.

[For86] S. Fortune, A sweepline algorithm for Voronoi diagrams, *Proceedings of the Second Annual Symposium on Computational Geometry*, 1986, 313-322.

[GKL83] I. G. Gowda, D. G. Kirkpatrick, D. T. Lee and A. Naamad, Dynamic Voronoi Diagrams, *IEEE Transactions on Information Theory 29*,5 (1983), 724-731.

[GuS85] L. J. Guibas and J. Stolfi, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, *ACM Transactions on Graphics 4*(1985), 74-123.

[Mar82] D. Marr, *Vision*, W.H. Freeman and Co., 1982.

[MaA78] W. N. Martin and J. K. Aggarwal, Survey: Dynamic Scene Analysis, *Computer Vision, Graphics and Image Processing 7*(1978).

[MaA88] W. N. Martin and J. K. Aggarwal, *MOTION UNDERSTANDING: Robot and Human Vision*, Kluwer Academic Publishers, 1988.

[OvL81] M. H. Overmars and J. Leeuwen, Maintenance of Configurations in the Plane, *J. Computer and System Sciences 23*(1981), 166-204.

[PrS85] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[RiS89] D. Richards and J. S. Salowe, A Linear-Time Algorithm to Construct a Rectilinear Steiner Minimal Tree for k-Extremal Point Sets, presented at the *NATO Advanced Workshop on Topological Network Design*, 1989.

[Ric89] D. S. Richards, Fast Heuristic Algorithms for Rectilinear Steiner Trees, *Algorithmica 4*(1989), 191-207.

[Sei86] R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, *Proceedings of the Eighteenth Symposium on the Theory of Computing*, 1986, 404-413.

[TaM86] C. L. Tan and W. N. Martin, A Distributed System for Analyzing Time-varying Multiresolution Imagery, *Computer Vision, Graphics, and Image Processing 36*(1986).

[TaM89] C. L. Tan and W. N. Martin, An Analysis of a Distributed Multiresolution Vision System, *Pattern Recognition 22*(1989).

F.6 Impact of the Infrastructure Enhancement

An excellent infrastructure is a necessary requirement for excellent experimental research. Quality computation resources need to be readily accessible and present in sufficient quantity. A researcher needs to be able to design an experiment and find the infrastructure supportive of it. This means that the researcher will invest most of his intellectual effort in the research problem, not the mechanics of experimentation. In this proposal we have outlined how we will use NSF resources to improve our experimental infrastructure. *There are many impacts.*

Students will gain ready, convenient assess to computation resources. Today, only 12 UNIX-based workstations are generally available to computer science students. Twenty-three of the 43 workstations are located in faculty and technical staff offices; 8 stations are in laboratories and their use is constrained in some way. Likewise, in electrical engineering 10 workstations are shared by 25 computer engineering graduate students and 60 graduate students each semester in classes on switching theory, computer design and VLSI system design. This is an average of one workstation for every 10 students. We are requesting some workstations, high performance servers and X-window viewports. Together, these will greatly improve accessibility and modestly increase cycles per student to be used both for research and graduate education.

Scarce resource bottlenecks will be relaxed. For example, the requested high speed raster plotter will reduce the elapsed time to plot a dense circuit from 8 hours to 5 minutes. Also, more computer-intensive jobs will execute faster in the areas of physical design, algorithm testing, circuit test pattern generation, WM benchmark simulation, and circuit simulation.

Selected research efforts will be accelerated. Equipment allocations are made to two research areas: parallel computation and networks. Parallel computation is a focus of research attention as evidenced by the presence of three substantial parallel engines. The BBN ''Butterfly'' arrived some months ago. We anticipate adding a new parallel engine roughly every 18 months to two years. This budget allocates funds for one new, to-be-determined parallel engine about 4 years from the arrival of the ''Butterfly''. The second area for research acceleration is networks. We expect a dramatic increase in communications speed, first in the research laboratory and then in our computation infrastructure. The proposed network funds are for research equipment. Research experience will chart the way to increase the LAN speeds for computer science and engineering and for our future collaborators in other departments in the University. The impact of this infrastructure is to let us gain experience early, as well as to do research.

Researchers will be able to prototype circuits and systems rapidly. The research descriptions in this chapter amply demonstrate that we design and build experimental circuits and systems (both hardware and software). To enhance this direction, we are proposing to add equipment to the current integrated circuit design facility within CSIS and to create a new system integration capability. Within CSIS, we are increasing the number of design stations, resulting in a 40% increase in the number of ''seats.'' In addition, a programmable logic device implementation capability will be purchased for the rapid prototyping of logic functions. The system integration facility, housed within the System Integration Laboratory will bridge between the software design efforts and the hardware design. Such an infrastructure addition is costly. It probably can not be built up incrementally, because a critical mass of knowledge and equipment is necessary to sustain it. Only programs like the NSF Infrastructure Program fund such advancements. We will be able to prototype hardware based on off-the-shelf technology and application-specific ICs. Then, we can develop several instruction set implementations for the WM architecture family and WM multicomputer prototype. For networks we can integrate protocol chips into an experimental interface microprocessor. For the first time we will be able to lash-up laboratory board systems to our parallel engines, perhaps to test proposed parallel operators or better monitor data access traffic.

Secondly, the SIL will greatly extend our options for experimenting with software that is ''close to'' the hardware/software interface. Many of us build such software, for example real-time data bases, real-time operating systems, communication interfaces and fault-tolerance software. To date, we experiment only by adapting the software. With the SIL we have some of the capability to experiment with the hardware side of the interface.

Lastly, this infrastructure enhancement will strengthen the synergistic relationship between the computer science and electrical engineering faculty and students. It will catalyze the research projects and the winning of other grants.

A simple quantitative measure of improvement directly attributable to the infrastructure is the amount of time savings that can be experienced for the completion of a project. We believe that in several cases, a realistic 25-35 percent of the duration of a research project could be saved with no loss of results.

C. Equipment

The Computer Science department currently operates the following equipment:

VAX 8600 running 4.3 BSD Unix with 72 Mbytes of primary memory and 1.6 Gbytes of local disk space
Ethernet-based LAN with two SUN 3/280 servers having 4.4 gigabtyes disk capacity
Workstations: 36 SUN 3's (23 with local disks), one SUN 4/110, 6 SUN 2's, Concurrent 3230 minicomputer
Parallel engines: 32 Node Intel Hypercube with 1.4 gigabyte disk farm and 4 Mbytes of memory; 128 Node NCUBE with 8x160 megabyte disk farm; 4 386-based NCUBE-4 systems, 1 AT-based NCUBE-4 system; BBN Butterfly with 32 Nodes sharing 128 Mbytes of primary memory and 500 Mbytes of disk
Personal computers: 10 PC AT class machines, 6 Intel 386 machines, 3 Macintosh IIs, 2 VAXstation GPX systems
Viewports: 15 X-window terminals, 24 'dumb' terminals
Printing: 6 Apple Laserwriter printers

The Center for Semicustom Integrated Systems currently operates the following equipment:

Minicomputers: Harris HCX-9 Super Minicomputer, 2 VAX-11/750 computers running Unix and VMS
Workstations: 8 Sun 3/50 and 3/60 workstations, 7 Apollo 3000 & 3500 workstations
Viewports: 4 AED 512 color graphics units
Personal computers: 10 IBM PC/AT computers
Test Generation/Application: Tektronix DAS 9100 tester
Plotting: 2 Hewlett Packard multipen plotters

TCP/IP networking connects all locations on University grounds, running on a 10 Mbps Ethernet.

The equipment budget follows the equipment rationale section.

Equipment Rationale

Our requested budget for equipment is approximately $2.2 million. We will invest 38% in general purpose computing facilities, 13% in enhancing the integrated circuit design facilities within the Center for Semicustom Integrated Systems (CSIS), 19% in creating the Systems Integration Laboratory (SIL), 9% in support for parallel computation, 7% in high-speed networking, and 14% in software tools for all the above areas.

The equipment budget is presented in terms of currently-available products at current prices. Actual hardware and software purchased will be the most up-to-date items available at the time of purchase. Discounts are noted.

Center for Semicustom Integrated Systems: The CSIS houses and maintains the facilities for the design, fabrication using MOSIS, and test of integrated circuits. Capabilities within the

CSIS include design capture and functional simulation, performance modeling and simulation, test pattern generation and fault simulation, physical design and synthesis, and hardware test. We require four improvements: (1) additional commercial computer-aided design software for performance modeling (ADAS and SES/workbench), automated synthesis (Silicon Compiler Systems), and programmable logic device design (Data I/O Personal Silicon Foundry); (2) enhancement of hardware test moving us from small, slow devices (25 megahertz and 64 pins) to higher performance devices (100 megahertz and 128 pins); (3) a new probe capability for minimal die-level testing; and (4) a 40 per cent increase in laboratory design seats giving us a total of 10 seats. In addition, the latter requires an increase in general computing to support our core tool set (Mentor Graphics) using Apollo DN10000 and DN3500 technology. We are committed to using Mentor Graphics tools. The enhanced facility will provide the complete rapid prototyping capability for both custom IC and "glue" logic implementations. The high performance compute server will provide the needed resources for excellent design cycle times.

Systems Integration Laboratory: This Laboratory will support the fabrication of systems from off-the-shelf components, custom circuits designed in the IC Design Laboratory, standard digital and analog parts, and programmable logic devices. No such facility currently exists. It will be constituted using all appropriate software in the CSIS, specifically the Mentor Graphics tool set, plus tools to support printed circuit board layout with automated wiring. Hardware requested includes three color workstations (DN3500s) and five fabrication/test stations. The latter are equipped with power supplies and function generators and share low- and high-frequency oscilloscopes and logic analyzers. This laboratory will be networked at high speed to the CSIS as well as to all general purpose CS and EE computing. Interfaces to commercial printed circuit board fabrication houses will be established to provide quick implementation turnaround. In addition, multichip and single chip microprocessor development capabilities will be installed. Down-loading cross compilers and assemblers will be routinely available. The overall objective of the SIL is to provide the resources and expertise necessary on a continuing basis to support easy and quick board/system-level prototyping.

General Computing Environment: We seek to improve our general computing environment. Our research is computation-intensive. We need for high performance workstations, file servers, and peripherals such as color graphics monitors, color plotters, and color printers. We intend to provide access to the computation cycles via the X-window viewports. All of the computational equipment requested will support X-windows which will permit researchers to access any of our machines from anywhere on the network. Graphics workstations are also requested; the VLSI physical design work with detailed routers and switchbox routers (Section F.1.1) will profit enormously from the addition of advanced graphics equipment.

Network Equipment: All of the computing equipment will be networked with a combination of Ethernets and high-speed fiber optic LANs. We will use the 100 Mbit/sec Fiber Distributed Data Interface (FDDI) as a testbed for our real-time communications protocols and as a high-speed access path between equipment located in Computer Science and equipment located in Electrical Engineering. The design and implementation of our own real-time transport protocols and protocol chip sets (section F.3.2) requires LAN capacities of 100 Mbit/sec and higher.

Parallel Computing Equipment: Our Institute for Parallel Computation (IPC) currently operates a Hypercube, N-cube, and Butterfly. Current research projects will utilize existing equipment through budget year three, at which time a replacement for some of this equipment will be required. The appropriate architecture and vendor will be chosen at the time of purchase.

The IPC provides parallel computational support for the work in parallel databases, parallel simulation and parallel genetic algorithms (see Section F.4). The IPC supports other research not discussed in this proposal.

Software Tool Sets: Many capabilities are realized as sets of software tools. In many cases groups of tools need to be integrated to work in concert, or at least fused with user-convenient interfaces. We list software tool areas below. Example software systems are included, however particular choices will vary based on actual negotiated cost and current technology. The budget contains a yearly line item for software license fees and maintenance.

Integrated software development environment: Compilers or cross-compilers for all machines in the environment. C and C++ are the preferred languages. We also need tools for source control, configuration control, structured analysis, debugging, and regression testing.

Software analysis: Performance profiler and analysis support for parallel programming.

Simulation and modelling: SIMSCRIPT, ADAS, event simulation debugger.

Fault tolerance analysis: CARE-III, HARP.

User Interfaces: X-windows base plus tailored windows for a variety of applications.

Information management: Hypertext/database/information retrieval support. The tools should integrate access to text, programs, images, and possibly audio.

VLSI circuit design and analysis: Mentor Graphics tool suite.

Document production: We now use troff and related tools and wish to move to a WYSIWYG document system with an integrated database interface for bibliographic information.

Presentation graphics tools: Slide packages and an algorithm animation capability. We need a tool that supports the integration of visual aids into program documentation.

Maintenance

Equipment maintenance costs are computed yearly as eight per cent of accumulated hardware acquisition costs. Note that costs reflect discounts. The software budget includes its own maintenance.