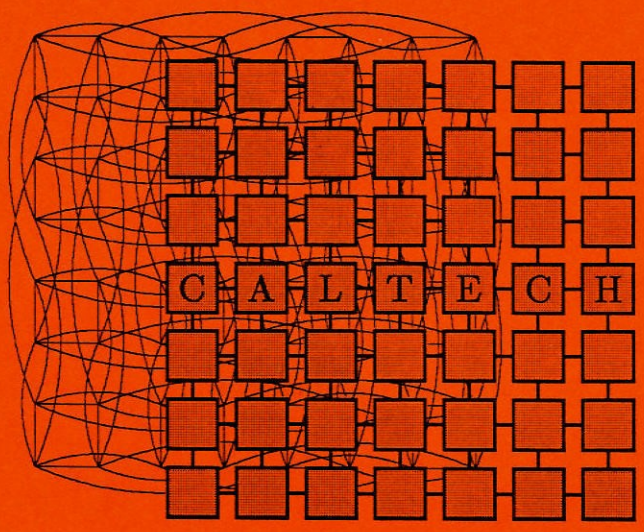

SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-88-18

9 November 1988

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-88-18

9 November 1988

Reporting Period:	1 April 1988 – 31 October 1988 (7 months)
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	William C. Athas
	K. Mani Chandy
	Alain J. Martin
	Martin Rem
	Charles L. Seitz
	Stephen Taylor

Sponsored by the
Defense Advanced Research Projects Agency
DARPA Order Number 6202

Monitored by the
Office of Naval Research
Contract Number N00014-87-K-0745

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of the research activities and results for the seven-month period, 1 April 1988 to 31 October 1988, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

1.3 Changes in Key Personnel

Dr. William C. Athas completed his appointment as a Postdoctoral Research Fellow in Computer Science in August 1988, and has joined the faculty at the University of Texas at Austin as an Assistant Professor of Computer Science. Dr. Stephen Taylor, a new PhD from the Weizmann Institute of Science and the author of a multicomputer implementation of flat concurrent prolog, joined the project in September 1988 with an appointment at Caltech as an Instructor in Computer Science.

2. Architecture Experiments

2.1 Mosaic Project

Bill Athas, Charles Flaig, Glenn Lewis, Jakov Seizovic, Don Speck, Wen-King Su, Tony Wittry, Chuck Seitz

The Mosaic C is an experimental multicomputer with single-chip nodes, currently in development. The stipulation that the nodes fit on a single chip so limits the storage for each node that relatively fine-grain concurrent programming techniques must be used. The Mosaic C will be programmed using the Cantor programming language, a fine-grain object-based (or Actor) language. We are working toward building a 16K-node Mosaic system using nodes fabricated in 1.2μm CMOS technology, with a near-term milestone of a 1K-node system using nodes fabricated in 1.6μm CMOS.

Much of our effort in this period has been concentrated on the Mosaic C project. The following is a brief summary of these activities (See also sections 3.1 & 4.5):

1. Cantor version 2.2 has been used internally within the research group for the past several months, and has been documented for external distribution. A technical report describing a collection of exemplary Cantor 2.2 programs that range up to 15 pages of program text in length was published. The report also reports the rationale for many of the design decisions in the evolution of Cantor from version 2.0 to 2.2.

2. Our initial implementation of a Cantor code generator for the Mosaic C indicated that only a simple procedure call mechanism was required; otherwise, the Mosaic C instruction set has been an efficient target for code generation. Work has commenced on a final Cantor code generator and runtime system for the Mosaic.

3. In accordance with the studies of code generation, the microcode for the Mosaic C processor was revised to implement an instruction set having a simpler procedure-call mechanism, together with several other minor refinements. The simplification of the instruction set reduced the number of implicants in the microcode that controls the processor from 66 to 102. The impact of this simplification on the processor area is merely favorable; its greatest benefit is in improving the processor speed (the RISC effect).

4. The entire processor was simulated at the clock-cycle and microcode level to debug and verify the microcode. The verified microcode was then used to generate a PLA structure, which was tied to the Mosaic C datapath for switch-level simulation and verification of the entire processor. A hybrid static/precharge PLA was designed to maximize the performance, and will be used in the final version of the processor.

5. An interface between the router and memory was designed, laid out, and verified by switch-level simulation. This final section of the Mosaic C single-

chip multicomputer node also includes the arbitration for memory refresh and memory access.

Fabrication of the first prototype processors and full Mosaic elements is now anticipated for early CY1989.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Alain Martin, Bill Athas, Charles Flaig, Jakov Seizovic, Craig Steele, Wen-King Su

Deliveries of the first production models of the Ametek Series 2010, a second-generation medium-grain multicomputer developed as a joint project between our research project and Ametek Computer Research Division, took place in this period. The reports we have received have been favorable. One customer who is also a DARPA contractor had developed 10,000+ lines of source code using the Cosmic Environment prior to taking delivery of the Ametek 2010, and apparently ported this code in a few days with no difficulties.

Additional benchmarks on the Ametek Series 2010 continue to show that it runs 8–10 times faster per node than such first-generation machines as the Intel iPSC/1.

Copies of the Cosmic Environment system have been distributed to approximately an additional 35 sites in this period, bringing the total copies distributed directly from the project to over 150. In addition, source copies of the Reactive Kernel node operating system were provided to two government contractors who are purchasing Ametek 2010 systems. An article titled "Multicomputers: Message-Passing Concurrent Computers" was published in the August 1988 issue of *IEEE COMPUTER*. This article on the current status of the multicomputers that have developed out of the work of our research group stimulated requests for many additional copies of "The C Programmer's Abbreviated Guide to Multicomputer Programming" [Caltech-CS-TR-88-1].

We expect to take delivery of the first 16-node increment of a 256-node Ametek 2010 in November 1988, and also a 16-node Intel iPSC/2, which will later be expanded to 64 nodes. Substantial blocks of time on the Ametek 2010 will be available to guest DARPA researchers.

Our Caltech project continues to work with both Ametek and Intel on the architectural design, message-routing methods and chips, and system software (evolutions of the Reactive Kernel (RK) node operating system and the Cosmic Environment (CE) host runtime system) for multicomputers. (See sections 3.2, 3.6 and 4.6 for details on these efforts.) We expect to see additional major advances in the performance and programmability of these systems over the next two years. In

* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Ametek Computer Research Division (Monrovia, California).

addition, we continue to develop applications in VLSI design and analysis tools, and in other areas in which the programming of these multicomputer systems presents particular difficulties or opportunities. (See sections 3.3–3.5 and 4.9.)

2.3 Cosmic Cube Project

Bill Athas, Wen-King Su, Jakov Seizovic, Chuck Seitz

This section summarizes the current usage and the hardware and software status of our first-generation multicomputers, the Cosmic Cubes and Intel iPSC/1 d7.

These systems continue to operate reliably. Overall usage has been moderately heavy. The most time-consuming application in this period from within our own group has been a continuation of an extensive series of simulations by John Ngai concerned with the maximal utilization of networks with faulty routers or channels (see section ?). Supersonic flow computations being performed by students and faculty in Aeronautics at Caltech continue as the largest share of outside use.

The 64-node Cosmic Cube exhibited a hard failure in this seven-month period, a complete failure of its primary 5V, 130A power supply. The power supply was replaced, and the system rebooted without any problems. Counting the power supply failure as a single failure, the two original Cosmic Cubes have now logged 3.6 million node-hours with only four hard failures, three of them being chip failures in nodes. Curiously, we have not encountered a single connector failure. The calculated node MTBF of 100,000 hours reported before these machines were constructed was extremely conservative. A node MTBF in excess of 1,000,000 hours is probable, and can be stated at a 54% confidence level.

Our Intel iPSC/1 d7 (128 nodes) was contributed to the Submicron Systems Architecture Project as a part of the license agreement between the Caltech and Intel, and is accessible via the ARPAnet to other DARPA researchers who may wish to experiment with it. To request an account, please contact `chuck@vlsi.caltech.edu`. The Ametek Series 2010 system to be installed later this month will be available for outside use on a similar basis.

3. Concurrent Computation

3.1 Cantor

Nanette J. Boden, William C. Athas, Chuck Seitz

Programming for Fine-Grain Multicomputers

Over the last year we have been conducting a series of fine-grain programming experiments using Cantor. The purpose of this series of experiments was both to evaluate Cantor as a programming language and to investigate the nature of fine-grain programming. Application programs that have been written in these experiments include: fast-Fourier transform, shortest-path algorithms, a 2D convex hull solver, R-C chain-circuit simulation, digital logic simulation, a checkmate analyzer, an enumerator of paraffin isomers, and many others.

As a result of these programming experiments, modifications to Cantor have been made to facilitate fine-grain programming. Iteration internal to objects, custom objects, functional abstraction, and one-dimensional vectors are programming constructs that are now available in the newest version of Cantor, Cantor 2.2. A feature has also been added to the language to permit rudimentary discretion over message receipts. Analysis of the programming experiments clearly indicates that programming situations exist where some message discretion is very useful. In addition to these modifications, unnecessary features of the original language specification have been removed, including dynamic typing of variables. The changes that have been made to Cantor thus enhance programming abstraction while removing unnecessary constructs.

Using the latest version of Cantor as an experimental tool, we have written enough programs in the fine-grain style to draw some conclusions. Although formulations for Cantor programs are myriad, we have detected three general paradigms for the development of fine-grain programs:

1. Functional program specifications can be mapped directly into message-driven programs.
2. Solution specifications can be mapped into message-driven programs.
3. The object program can operate as a "logical apparatus" to solve the application problem.

In addition to observing these paradigms, we have been encouraged by the high degree of concurrency that is achieved in Cantor programs and by the convenience and generality of fine-grain programming. Based on our experiments with Cantor thus far, we believe that large, highly concurrent programs can be efficiently expressed in the fine-grain programming style.

Programming for the Mosaic

Recent research in the area of Mosaic programming has focused on the definition and analysis of an abstract machine for the execution of Cantor code. The Cantor Abstract Machine (CAM) definition is based on the fine-grain multicomputer architecture, yet encapsulates operations like object creation, message sends and receives, *etc,* in single instructions. The purpose of this approach is to isolate the implementation of these complicated operations as much as possible from the development of an efficient runtime system.

A new Cantor code generator and simulator have been written for the CAM. Analysis of the abstract machine has already suggested improvements in the Cantor intermediate format. In addition, simulation of program execution on the CAM is expected to be very useful in evaluating potential Mosaic runtime system alternatives.

3.2 The Cosmic Environment and Reactive Kernel

Jakov Seizovic, Wen-King Su, Chuck Seitz

The Cosmic Environment and Reactive Kernel continue to run reliably on the original Cosmic Cubes and on the Ametek Series 2010, and no major changes have been made. The internals of RK are now documented in technical report Caltech-CS-TR-88-10.

In the original version of the RK, we were able to guarantee the *weak fairness* of scheduling on a multicomputer node only if all processes on that node satisfied the reactive property that they would eventually either terminate, or execute an `xrecv()`. The producers of an infinite number of messages are an important class of processes that do not satisfy the reactive property. A simple modification of the implementation of the `xmalloc()` system call has enabled us to support the infinite computations as well. The `xmalloc()` system call is implemented in terms of the RPC mechanism. The requested buffer is not delivered immediately; instead it is sent to the requesting process and delivered through the regular scheduling mechanism.

3.3 CONCISE — A Concurrent Circuit Simulator*

Sven Mattisson, Lena Peterson, Chuck Seitz

Within this project, a concurrent circuit simulation program called CONCISE has been developed. This program is a circuit simulator for transient analysis of CMOS-circuits. It is written in C and uses the Cosmic Environment/Reactive Kernel message-passing primitives.

* This segment of our research is a joint project between the Caltech Submicron Systems Architecture Project and the Department of Applied Electronics at the University of Lund, Sweden.

Recently, CONCISE was ported to the Ametek Series 2010. Thus, the program now runs on several multicomputers with loosely coupled nodes, including the Ametek 2010 and the Intel iPSC, and on a shared memory multicomputer, the Sequent Symmetry. The port to the Ametek 2010 showed that CONCISE is more than eight times faster on the Ametek 2010 than on the Intel iPSC/1, which is a typical first-generation multicomputer.

The Reactive Kernel primitives support a programming model where each process has its own memory space. This model makes dynamic partitioning and load balancing expensive in CPU time. Thus, we have developed a static partitioning scheme that tries to enhance the convergence rate of the waveform relaxation method without sacrificing the grain-size of the computational tasks. It is important to notice that the requirements on the partitioning algorithms in this case differ from the "traditional" parallelization, where only a few processing nodes are used.

So far, six different combinations of iteration schemes and partitioning have been tested. The iteration schemes tested are ordinary Jacobi iterations, ordinary Gauss-Seidel, and n-colored Gauss-Seidel. The n-colored Gauss-Seidel uses the incidence-degree algorithm to find a coloring with the least number of colors for the circuit graph. Then, the different colors can be solved concurrently, since each node has a color different from those of its neighbors. These three algorithms have all been run with two different partitioning schemes: one in which each circuit node forms a cluster on its own, and one where source-drain connected circuit nodes are clustered together.

The results show that regular Gauss-Seidel iterations are not suitable except for very few processing nodes, and this scheme is the most popular for sequential waveform-relaxation implementations. Instead, the n-coloring version of Gauss-Seidel iterations are useful for the case when the number of processing nodes is large, but significantly less than the number of processes. The number of colors needed usually lies between three and five.

When the number of computing nodes is close to the number of circuit nodes, Jacobi iterations do surprisingly well. This is due to the fact that the load imbalance gets increasingly severe for the other schemes. For some circuits, the clusters get very big, and splitting schemes fail in producing reasonable size clusters that still achieve comparable convergence speed. For such circuits a hierarchical approach where more than one node can be assigned to solving a cluster would be desirable. Such an approach will be possible with the faster message passing of the second-generation multicomputers, and experiments in this area are presently being carried out.

In another effort, Concise has been used by Anthony Skjellum in the Chemical Engineering Department at Caltech for the simulation of distillation columns. This work has shown that it is possible to use Concise to simulate dynamic systems that

are not at all like circuits. As part of this effort, Concise has been modified to make it easier to install models of other kinds of "devices."

3.4 Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm

Wen-King Su, Chuck Seitz

A new and more versatile logic simulator has been written in the past six months to better evaluate a more diverse set of conservative variants of the Chandy-Misra-Bryant (CMB) distributed discrete-event simulation algorithm. Most of the conclusions from this study are included in the paper "Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm," accepted for publication in the 1989 SCS Eastern Multi-conference. The primary conclusions are that the variants examined are similar, in that all of them take an initial penalty running on a single node in comparison with sequential event-driven simulators that exploit an ordered event list. The penalty is due to the generation and the processing of null messages. However, as the number of processing nodes increases, the simulation time decreases linearly until all usable concurrency has been exhausted. Depending on the circuit being simulated, the crossover point (the point at which the time taken by the concurrent simulators drops below the time taken for the sequential simulator) has been observed to be anywhere between four and 200 nodes.

After the paper was submitted, a new simulator variant was written to try to reduce the initial overhead by combining sequential simulation methods with the concurrent simulator variants. The resulting simulator has the performace of a sequential simulator for the single processor case, and it converges with that of the concurrent simulator when the number of nodes is sufficiently large. However, the nature of the logic circuit being simulated strongly influences the rate of convergence. We have observed all three cases:

1. The simulation time humps upward toward that of the concurrent simulators as soon as the number of processing nodes is increased beyond one.
2. The simulation time remains the same until the concurrent-sequential crossover point.
3. The simulation time starts to decrease as soon as the number of nodes are increased, but the drop is less than linear.

A conclusion of this study is that very-high-performance logic simulation on concurrent computers is completely plausible for systems with very large numbers of nodes, where the CMB null-message scheme is fully exploited. Conversely, it is efficient for small-N systems only when the elements being simulated are more complex and have longer running times than logic elements.

3.5 Automatic Mapping of Processes and Channels

Drazen Borkovic, Alain Martin

To facilitate programming of message-passing machines, we have developed a preprocessor, map[3], that allows for a certain level of abstraction in the mapping of processes and channels on the nodes and physical channels of a message-passing multicomputer.

The description of a set of processes and the channels between them has been compiled into a set of C functions that perform the mapping of the processes onto physical nodes of the target machine. The preprocessor supports a hierarchical organization of processes and local names for the channels. There is also a set of library routines that can emulate channels with arbitrary slack.

The preprocessor and the library routines have been successfully implemented and tested under the Cosmic Environment/Reactive Kernel system.

3.6 A Multicomputer "Page Kernel"

Craig S. Steele, Chuck Seitz

As described in a previous report, an experimental "page kernel" is being developed that uses memory-access-protection mechanisms as the interface to multicomputer message subsystems. A prototype of the "page kernel" is now running on a sequential machine. The current code is simulating the memory-management hardware of the Ametek Series 2010 computing node, and will be ported to the Series 2010 shortly.

The page kernel supports dynamic load-balancing and process relocation. The kernel's ability to transparently update copies of data distributed across a multi-node system is particularly well-suited for chaotic iterative programs, such as process-placement optimization.

4. VLSI Design

4.1 Testing Self-Timed Circuits

Pieter Hazewindus, Alain Martin

We are investigating methods to test self-timed circuits. Traditionally, it is thought that these circuits are hard to test because of the possibility of races and hazards, and because these circuits are sequential. In our design method, however, races and hazards are absent.

The fault model we use is the stuck-at model, where each wire may be stuck forever at a high (logic-1) or low (logic-0) voltage. We have proven that it is sufficient to perform a single four-phase handshake on each channel to detect all detectable stuck-at faults. Some faults are undetectable.

For the automatic compilation, the main sequencing element is the so-called D-element. For the D-element, there are twenty-two possible stuck-at faults, two of which are undetectable. We have designed an alternate D-element that does not have any undetectable stuck-at faults. Most other circuit constructs in this compiler are completely testable.

Although it is not yet certain whether all constructs can be made entirely testable, our present estimate is that self-timed circuits designed according to our method should be easier to test than traditional clocked circuits.

4.2 A Self-Timed $3x + 1$ Engine

Tony Lee, Alain Martin

We have designed and fabricated a self-timed special-purpose processor for implementing the $3x+1$ algorithm. The processor consists of a state-machine and an 80-bit-wide datapath. It contains approximately 40,000 transistors and operates at over 8 MIPS in 2μm MOSIS SCMOS technology. As usual, the chip was functional on first silicon.

4.3 Performance Analysis of Self-Timed Circuits

Steve Burns, Alain Martin

We have developed methods for determining the repetition time of a set of communicating sequential processes described as handshaking expansions. This performance measure is provided in the form of constraint equations involving symbolic values of the communication and sequencing delays. The analysis is valid regardless of the actual delay values, and thus provides a means of comparing designs described at the handshaking expansion level without first generating detailed circuit implementations. Circuits for handshaking expansions that result in slow repetition times need never be designed.

This method has proven particularly useful in the analysis of programs involving data. It has been used throughout the design of the self-timed microprocessor, increasing the performance of programs involving data up to a factor of two.

4.4 The Design of a Self-Timed Microprocessor

Alain Martin, Steve Burns, Tony Lee, Drazen Borkovic, Pieter Hazewindus

In order to refute the claims that our design method would be too slow and too wasteful in area for anything but small circuits, we have embarked on the design of complete general-purpose microprocessor. The instruction set is "classic": 16-bit instructions with offset, load/store type of instructions, and separate memories for instructions and data. The only restriction is the absence of an interrupt mechanism.

As expected, since the method is based on concurrent programming techniques, the design is highly concurrent. The fetch, decode, and execute phases overlap, as do the execution of ALU and memory instructions. The different processes share 16 general-purpose registers, and four buses are used to communicate with the registers, in addition to point-to-point channels.

We are now in the layout phase of the design. Preliminary estimates of the performance are encouraging. In 2μm SCMOS, we expect to reach 20MIPS.

4.5 Mosaic Elements

Chuck Seitz, Bill Athas, Charles Flaig, Glenn Lewis, Don Speck, Jakov Seizovic, Wen-King Su

With the completion of the packet interface section and the near-completion of the processor, and with the other sections having already been fabricated and tested, the Mosaic C single-chip multicomputer node is rapidly approaching completion. Assembly of the sections will start within the next month, and fabrication of complete elements early in 1989.

The packet interface for the Mosaic chip has been layed out and verified with the switch-level simulation. It is entirely synchronous, and was designed conservatively, so no problems with it are anticipated.

The packet interface consists of two independent finite-state machines, one for sending packets, and the other for receiving packets. Both machines act as simple DMA channels, stealing unused memory cycles, and the packet interface is designed to be able to sustain a throughput equal to the maximum possible message rate that can be achieved by the message router.

The packet interface provides for a fairly complete testing of itself and the router, initiated by a CPU request to send a message to itself. In this mode of operation, the message will be taken from the memory, sent through all three router dimensions, and received back into the memory.

4.6 Fast Self-Timed Mesh Routing Chips

Charles Flaig, Chuck Seitz

A new design of a mesh routing chip (MRC), the FMRC2.0 design, was sent to fabrication in May 1988, together with a separate test chip containing only the FIFO used in the FMRC2.0. These chips employ a circuit design style that is potentially faster but less conservative than is usual for self-timed designs. The chips returned from fabrication do indeed operate nearly three times faster than previous designs. The FIFO test chip, fabricated in a 2μm MOSIS SCMOS process (this chip was also a test of the new 40-pin 2μm pads and design frame that we developed for MOSIS) operated correctly at 70 MBytes/s!

The critical path in a routing chip includes somewhat longer delay paths due to the switching of the packets; hence, although the FMRC2.0 was fabricated in a 1.6μm process, and its FIFOs might be expected to operate at around 85 MBytes/s, it operates as anticipated at 70 MBytes/s. However, it routes packets incorrectly, showing symptoms of directing packets according to the tail of the previous packet rather than the head of the current packet. This fault was finally traced to a timing error of approximately 0.7ns in the latching of a routing decision. The timing error was fixed, and the timing margins in the entire chip were reexamined. A *post facto* Spice simulation of what the analysis showed were the critical points in the old and new designs verified that the original design had a timing error of 0.7ns, while the revised design has a timing margin of about 1.0ns (about 50% of the difference between two short delay paths; hence, not as close as it may sound).

If successful, we expect this new FMRC chip to replace the MRC currently used in the Ametek Series 2010 multicomputer. With help from George Lewicki, this design is also being transferred to an Intel fabrication process for possible use in a future Intel multicomputer.

Tests of the self-timed FIFO in a 2μm MOSIS SCMOS technology will be of interest to other chip designers in the DARPA VLSI community — particularly those designing self-timed chips.

The 2μm FIFO tests yielded a request $\rightarrow$ acknowledge time of 6.5-7.0ns, and a throughput of over 70 MBytes/s on these byte-wide channels. Lest someone interpret this test result as implying that we are driving 70MHz signals through these pads, please understand that in 2-cycle R/A signaling (*cf,* Mead & Conway, figure 7.16), only one *transition* is required for each data transfer, so the maximum fundamental frequency on any R/A or data pin is 35MHz to transfer data at a 70MHz rate.

The total fall-through time for all 101 FIFO stages was measured as 350ns, or 3.5ns fallthrough per stage. The fallthrough time calculated by the τ-model is about 70τ, so this is consistent with a value of τ for the 2μm MOSIS SCMOS n-well process of about 50ps (which is a bit smaller than expected). The *internal*

cycle time when the operation is not impeded by signals passing through pads and package pins is about 180τ, or about 9ns, corresponding to an internal throughput rate of 114MHz.

These speeds in the 2μ MOSIS n-well SCMOS technology are, as expected, about twice as fast as a nearly identical test device fabricated in a 3μm MOSIS p-well SCMOS process. The fallthrough times are more difficult to measure in the 1.6μm FMRC2.0 chip, because of switching and address-decrementing logic in the FIFO pipeline. We can infer than the FIFO fall-through times are about 2.8ns per stage, corresponding to a τ of 40ps, and an internal throughput rate of about 140 MHz.

It is quite evident from these tests that we are able to achieve much higher internal speeds with self-timed and/or asynchronous designs than we know how to achieve with clocked designs.

4.7 Adaptive Routing in Multicomputer Networks

John Y. Ngai, Chuck Seitz

Our studies of adaptive routing in multicomputer networks are approaching a conclusion, and have been generally successful. We now believe that the Adaptive Cut-Through (ACT) routing scheme is capable of outperforming the existing highly evolved oblivious routing devices by a factor of about two in throughput, and have numerous other advantages in hot-spot throughput and fault-tolerance. A summary of the results of our investigations is attached at the end of this report.

What remains to be done to realize the advantages of the ACT routing scheme is to design a VLSI routing chip and/or a new routing section for the Mosaic C.

4.8 Pads and Pad Frame Generation

Charles Flaig, Chuck Seitz

Derived in large part from the pads and pad frames we have designed for mesh routing chips (MRCs), a variety of new pad circuits have been designed for the $\lambda = 0.6\mu$m, 0.8μm, and 1.0μm MOSIS SCMOS processes. One of these design variations was used to produce a new 2μm 40-pin "tiny-chip" frame for MOSIS, including input, Schmitt input, output, and tristate output pads. The unusual features of these pad designs include the use of longitudinal (bipolar) clamp transistors for static and overvoltage protection, and a variety of pad pitches.

We can now report some test results for the 2μm pads. This 40-pin pad frame was fabricated with a 101-stage self-timed FIFO from the FRMC2.0 design (see section 4.6), together with some output pads being driven directly from input pads.

Overvoltage clamping on the inputs clamps to 6V at 200mA, and 7V at 800mA, which is excellent. Undervoltage protection is about the same as above, BUT, at

about -500mA the chip appears to suffer latchup (if power is supplied). This is not a problem for normal static, where no Vdd is applied, but if an input does goes more than about 1V negative while power is applied, latchup may be induced.

For the Schmitt input pad, trigger voltages are 0.8V and 3.9V, for a 2.9V hysteresis. Inpad → Outpad delay is 1.5–2.0ns for no load, 2.0–2.5ns for a fanout of 1, and 2.5-3.5ns for a fan-out of 2. Rise/fall time is 3.5ns for no load, 4.5ns for a fanout of 1, and 6.5ns for a fan-out of 2. The output pads can sink about 30mA at 1.0V, or source about 30mA at 4.0V, under 5.0V operation. These characteristics are more than adequate for student projects.

4.9 The Notorious CIF-flogger Program

Glenn Lewis, Chuck Seitz

The CIF-flogger is a multicomputer program for flattening CIF files, rasterizing the geometry, and for performing parallel operations on the geometry in strips. It runs under the CE/RK system, and hence, on most available multicomputers, including the Ametek Series 2010.

The CIF-flogger currently supports simple bloat, shrink, and logical operations on the flattened geometry, and hence can perform most geometrical design-rule checks. It establishes connected component labeling and will eventually provide complete design-rule checking, well checks, and circuit extraction. Based on timings on the iPSC/1, CIF-flogger is expected to perform design rule checks for 100K-transistor chips in much less than 1s per rule on second-generation multicomputers.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports

23 August 1988

Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check drawn on a U.S. bank in U.S. currency, payable to CALTECH.

____CS-TR-88-17	$3.00	*Constrained Diffrential Optimization for Neural Networks*, Platt, John C and Alan H Barr
____CS-TR-88-16	$3.00	*Programming Parallel Computers*, Chandy, K. Mani
____CS-TR-88-15	$13.00	*Applications of Surface Networks to Sampling Problems in Computer Graphics*, PhD Thesis Von Herzen, Brian
____CS-TR-88-14	$2.00	*Syntax-directed Translation of Concurrent Programs into Self-timed Circuits* Burns, Steven M and Alain J Martin
____CS-TR-88-13	$2.00	*A Message-Passing Model for Highly Concurrent Computation*, Martin, Alain J
____CS-TR-88-12	$4.00	*A Comparison of Strict and Non-strict Semantics for Lists*, MS Thesis Burch, Jerry R
____CS-TR-88-07	$3.00	*The Hexagonal Resistive Network and the Circular Approximation*, Feinstein, David I
____CS-TR-88-06	$3.00	*Theorems on Computations of Distributed Systems*, Chandy, K Mani
____CS-TR-88-05	$3.00	*Submicron Systems Architecture* ARPA Semiannual Technical Report
____CS-TR-88-01	$3.00	*C Programmer's Abbreviated Guide to Multicomputer Programming*, Seitz, Charles, Jakov Seizovic and Wen-King Su
____5258:TR:88	$3.00	*Submicron Systems Architecture* ARPA Semiannual Technical Report
____5256:TR:87	$2.00	*Synthesis Method for Self-timed VLSI Circuits*, Martin, Alain current supply only: see *Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design*, 224-229, Oct'87
____5253:TR:88	$2.00	*Synthesis of Self-Timed Circuits by Program Transformation*, Burns, Steven M and Alain J Martin
____5251:TR:87	$2.00	*Conditional Knowledge as a Basis for Distributed Simulation*, Chandy, K. Mani and Jay Misra
____5250:TR:87	$10.00	*Images, Numerical Analysis of Singularities and Shock Filters*, PhD Thesis Rudin, Leonid Iakov
____5249:TR:87	$6.00	*Logic from Programming Language Semantics*, PhD Thesis Choo, Young-il
____5247:TR:87	$6.00	*VLSI Concurrent Computation for Music Synthesis*, PhD Thesis Wawrzynek, John
____5246:TR:87	$3.00	*Framework for Adaptive Routing* Ngai, John Y and Charles L. Seitz
____5244:TR:87	$3.00	*Multicomputers* Athas, William C and Charles L Seitz
____5243:TR:87	$5.00	*Resource-Bounded Category and Measure in Exponential Complexity Classes*, PhD Thesis Lutz, Jack H

____5242:TR:87 $8.00 *Fine Grain Concurrent Computations*, PhD Thesis
Athas, William C.

____5241:TR:87 $3.00 *VLSI Mesh Routing Systems*, MS Thesis
Flaig, Charles M

____5240:TR:87 $2.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5239:TR:87 $3.00 *Trace Theory and Systolic Computations*
Rem, Martin

____5238:TR:87 $7.00 *Incorporating Time in the New World of Computing System*, MS Thesis
Poh, Hean Lee

____5236:TR:86 $4.00 *Approach to Concurrent Semantics Using Complete Traces*, MS Thesis
Van Horn, Kevin S.

____5235:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5234:TR:86 $3.00 *High Performance Implementation of Prolog*
Newton, Michael O

____5233:TR:86 $3.00 *Some Results on Kolmogorov-Chaitin Complexity*, MS Thesis
Schweizer, David Lawrence

____5232:TR:86 $4.00 *Cantor User Report*
Athas, W.C. and C. L. Seitz

____5230:TR:86 $24.00 *Monte Carlo Methods for 2-D Compaction*, PhD Thesis
Mosteller, R.C.

____5229:TR:86 $4.00 *anaLOG - A Functional Simulator for VLSI Neural Systems*, MS Thesis
Lazzaro, John

____5228:TR:86 $3.00 *On Performance of k-ary n-cube Interconection Networks*,
Dally, Wm. J

____5227:TR:86 $18.00 *Parallel Execution Model for Logic Programming*, PhD Thesis
Li, Pey-yun Peggy

____5223:TR:86 $15.00 *Integrated Optical Motion Detection*, PhD Thesis
Tanner, John E.

____5221:TR:86 $3.00 *Sync Model: A Parallel Execution Method for Logic Programming*
Li, Pey-yun Peggy and Alain J. Martin
current supply only: see *Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86*

____5220:TR:86 $4.00 *Submicron Systems Architecture*
ARPA Semiannual Technical Report

____5215:TR:86 $2.00 *How to Get a Large Natural Language System into a Personal Computer*,
Thompson, Bozena H. and Frederick B. Thompson

____5214:TR:86 $2.00 *ASK is Transportable in Half a Dozen Ways*,
Thompson, Bozena H. and Frederick B. Thompson

____5212:TR:86 $2.00 *On Seitz' Arbiter*,
Martin, Alain J

____5210:TR:86 $2.00 *Compiling Communicating Processes into Delay-Insensitive VLSI Circuits*,
Martin, Alain
current supply only: see *Distributed Computing* v 1 no 4 (1986)

____5207:TR:86 $2.00 *Complete and Infinite Traces: A Descriptive Model of Computing Agents*,
van Horn, Kevin

____5205:TR:85 $2.00 *Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity*,
Schweizer, David and Yaser Abu-Mostafa

____5204:TR:85 $3.00 *An Inverse Limit Construction of a Domain of Infinite Lists*,
Choo, Young-Il

____5202:TR:85 $15.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

___5200:TR:85 $18.00 *ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation*, PhD thesis
Whelan, Dan

___5198:TR:85 $8.00 *Neural Networks, Pattern Recognition and Fingerprint Hallucination*, PhD thesis
Mjolsness, Eric

___5197:TR:85 $7.00 *Sequential Threshold Circuits*, MS thesis
Platt, John

___5195:TR:85 $3.00 *New Generalization of Dekker's Algorithm for Mutual Exclusion*,
Martin, Alain J
current supply only: see *Information Processing Letters*, **23**, 295-297 (1986)

___5194:TR:85 $5.00 *Sneptree - A Versatile Interconnection Network*,
Li, Pey-yun Peggy and Alain J Martin

___5193:TR:85 $2.00 *Delay-insensitive Fair Arbiter*
Martin, Alain J
current supply only: see *Distr Computing* **1**:226-234 (1986)

___5190:TR:85 $3.00 *Concurrency Algebra and Petri Nets*,
Choo, Young-il

___5189:TR:85 $10.00 *Hierarchical Composition of VLSI Circuits*, PhD Thesis
Whitney, Telle

___5185:TR:85 $11.00 *Combining Computation with Geometry*, PhD Thesis
Lien, Sheue-Ling

___5184:TR:85 $7.00 *Placement of Communicating Processes on Multiprocessor Networks*, MS Thesis
Steele, Craig

___5179:TR:85 $3.00 *Sampling Deformed, Intersecting Surfaces with Quadtrees*, MS Thesis,
Von Herzen, Brian P.

___5178:TR:85 $9.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

___5174:TR:85 $7.00 *Balanced Cube: A Concurrent Data Structure*,
Dally, William J and Charles L Seitz

___5172:TR:85 $6.00 *Combined Logical and Functional Programming Language*,
Newton, Michael

___5168:TR:84 $3.00 *Object Oriented Architecture*,
Dally, Bill and Jim Kajiya

___5165:TR:84 $4.00 *Customizing One's Own Interface Using English as Primary Language*,
Thompson, B H and Frederick B Thompson

___5164:TR:84 $13.00 *ASK French - A French Natural Language Syntax*, MS Thesis
Sanouillet, Remy

___5160:TR:84 $7.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

___5158:TR:84 $6.00 *VLSI Architecture for Sound Synthesis*,
Wawrzynek, John and Carver Mead

___5157:TR:84 $15.00 *Bit-Serial Reed-Solomon Decoders in VLSI*, PhD Thesis
Whiting, Douglas

___5147:TR:84 $4.00 *Networks of Machines for Distributed Recursive Computations*,
Martin, Alain and Jan van de Snepscheut

___5143:TR:84 $5.00 *General Interconnect Problem*, MS Thesis
Ngai, John

___5140:TR:84 $5.00 *Hierarchy of Graph Isomorphism Testing*, MS Thesis
Chen, Wen-Chi

___5139:TR:84 $4.00 *HEX: A Hierarchical Circuit Extractor*, MS Thesis
Oyang, Yen-Jen

___5137:TR:84 $7.00 *Dialogue Designing Dialogue System*, PhD Thesis
Ho, Tai-Ping

____5136:TR:84 $5.00 *Heterogeneous Data Base Access*, PhD Thesis
Papachristidis, Alex

____5135:TR:84 $7.00 *Toward Concurrent Arithmetic*, MS Thesis
Chiang, Chao-Lin

____5134:TR:84 $2.00 *Using Logic Programming for Compiling APL*, MS Thesis
Derby, Howard

____5133:TR:84 $13.00 *Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems*, PhD Thesis
Lin, Tzu-mu

____5132:TR:84 $10.00 *Switch Level Fault Simulation of MOS Digital Circuits*, MS Thesis
Schuster, Mike

____5129:TR:84 $5.00 *Design of the MOSAIC Processor*, MS Thesis
Lutz, Chris

____5128:TM:84 $3.00 *Linguistic Analysis of Natural Language Communication with Computers*,
Thompson, Bozena H

____5125:TR:84 $6.00 *Supermesh*, MS Thesis
Su, Wen-king

____5123:TR:84 $14.00 *Mossim Simulation Engine Architecture and Design*,
Dally, Bill

____5122:TR:84 $8.00 *Submicron Systems Architecture*,
ARPA Semiannual Technical Report

____5114:TM:84 $3.00 *ASK As Window to the World*,
Thompson, Bozena, and Fred Thompson

____5112:TR:83 $22.00 *Parallel Machines for Computer Graphics*, PhD Thesis
Ulner, Michael

____5106:TM:83 $1.00 *Ray Tracing Parametric Patches*,
Kajiya, James T

____5104:TR:83 $9.00 *Graph Model and the Embedding of MOS Circuits*, MS Thesis
Ng, Tak-Kwong

____5094:TR:83 $2.00 *Stochastic Estimation of Channel Routing Track Demand*,
Ngai, John

____5092:TM:83 $2.00 *Residue Arithmetic and VLSI*,
Chiang, Chao-Lin and Lennart Johnsson

____5091:TR:83 $2.00 *Race Detection in MOS Circuits by Ternary Simulation*,
Bryant, Randal E

____5090:TR:83 $9.00 *Space-Time Algorithms: Semantics and Methodology*, PhD Thesis
Chen, Marina Chien-mei

____5089:TR:83 $10.00 *Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits*,
Lin, Tzu-Mu and Carver A Mead

____5086:TR:83 $4.00 *VLSI Combinator Reduction Engine*, MS Thesis
Athas, William C Jr

____5082:TR:83 $10.00 *Hardware Support for Advanced Data Management Systems*, PhD Thesis
Neches, Philip

____5081:TR:83 $4.00 *RTsim - A Register Transfer Simulator*, MS Thesis
Lam, Jimmy
current supply only: see *Acta Informatica* **20**, 301-313, (1983)

____5074:TR:83 $10.00 *Robust Sentence Analysis and Habitability*,
Trawick, David

____5073:TR:83 $12.00 *Automated Performance Optimization of Custom Integrated Circuits*, PhD Thesis
Trimberger, Steve

____5065:TR:82 $3.00 *Switch Level Model and Simulator for MOS Digital Systems*,
Bryant, Randal E

____5054:TM:82 $3.00 *Introducing ASK, A Simple Knowledgeable System*, Conf on App'l Natural Language Processing
Thompson, Bozena H and Frederick B Thompson

____5051:TM:82 $2.00 *Knowledgeable Contexts for User Interaction*, Proc Nat'l Computer Conference
Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho

____5035:TR:82 $9.00 *Type Inference in a Declarationless, Object-Oriented Language*, MS Thesis
Holstege, Eric

____5034:TR:82 $12.00 *Hybrid Processing*, PhD Thesis
Carroll, Chris

____5033:TR:82 $4.00 *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*,
Schuster, Mike, Randal Bryant and Doug Whiting

____5029:TM:82 $4.00 *POOH User's Manual*,
Whitney, Telle

____5018:TM:82 $2.00 *Filtering High Quality Text for Display on Raster Scan Devices*,
Kajiya, Jim and Mike Ullner

____5017:TM:82 $2.00 *Ray Tracing Parametric Patches*,
Kajiya, Jim

____5015:TR:82 $15.00 *VLSI Computational Structures Applied to Fingerprint Image Analysis*,
Megdal, Barry

____5014:TR:82 $15.00 *Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, PhD Thesis
Lang, Charles R Jr

____5012:TM:82 $2.00 *Switch-Level Modeling of MOS Digital Circuits*,
Bryant, Randal

____5000:TR:82 $6.00 *Self-Timed Chip Set for Multiprocessor Communication*, MS Thesis
Whiting, Douglas

____4684:TR:82 $3.00 *Characterization of Deadlock Free Resource Contentions*,
Chen, Marina, Martin Rem, and Ronald Graham

____4655:TR:81 $20.00 *Proc Second Caltech Conf on VLSI*,
Seitz, Charles, ed.

____3760:TR:80 $10.00 *Tree Machine: A Highly Concurrent Computing Environment*, PhD Thesis
Browning, Sally

____3759:TR:80 $10.00 *Homogeneous Machine*, PhD Thesis
Locanthi, Bart

____3710:TR:80 $10.00 *Understanding Hierarchical Design*, PhD Thesis
Rowson, James

____3340:TR:79 $26.00 *Proc. Caltech Conference on VLSI (1979)*,
Seitz, Charles, ed

____2276:TM:78 $12.00 *Language Processor and a Sample Language*,
Ayres, Ron

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

_____88-17	_____5238	_____5197	_____5135
_____88-16	_____5236	_____5195	_____5134
_____88-15	_____5235	_____5194	_____5133
_____88-14	_____5234	_____5193	_____5132
_____88-13	_____5233	_____5190	_____5129
_____88-12	_____5232	_____5189	_____5128
_____88-07	_____5231	_____5185	_____5125
_____88-06	_____5230	_____5184	_____5123
_____88-05	_____5229	_____5179	_____5122
_____88-01	_____5228	_____5178	_____5114
_____5258	_____5227	_____5174	_____5112
_____5256	_____5223	_____5172	_____5106
_____5253	_____5221	_____5168	_____5104
_____5251	_____5220	_____5165	_____5094
_____5250	_____5215	_____5164	_____5092
_____5249	_____5214	_____5160	_____5091
_____5247	_____5212	_____5158	_____5090
_____5246	_____5210	_____5157	_____5089
_____5244	_____5207	_____5147	_____5086
_____5243	_____5205	_____5143	_____5082
_____5242	_____5204	_____5140	_____5081
_____5241	_____5202	_____5139	_____5074
_____5240	_____5200	_____5137	_____5073
_____5239	_____5198	_____5136	_____5065

Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm

Wen-King Su and Charles L. Seitz
Department of Computer Science
California Institute of Technology

1. Introduction

We have been using variants of the Chandy-Misra-Bryant (CMB) distributed discrete-event simulation algorithm [1,2,3] since 1986 for a variety of simulation tasks [4]. The simulation programs run on multicomputers [5] (message-passing concurrent computers), such as the Cosmic Cube, Intel iPSC, and Ametek Series 2010. The excellent performance of these simulators led us to investigate a family of variants of the basic CMB algorithm, including lazy message-sending, demand-driven operation with backward demand messages, and adaptive adjustment of the parameters that control the laziness.

These studies were also motivated by our interest in scheduling strategies for reactive (message-driven) multiprocess programs [5,6,7], which are semantically similar to discrete-event (event-driven) simulators. The simulator itself is implemented in the reactive programming environment that we have developed for multicomputers, the Cosmic Environment, and the Reactive Kernel [8].

This paper is a brief and preliminary report of the simulation algorithms and performance results. A more definitive report will be found in the first author's forthcoming PhD thesis.

2. The CMB Simulation Framework

As usual, the system to be simulated is modeled as a set of communicating elements. A CMB simulator can be implemented by coding the behavior of elements in processes that communicate by messages. A message conveys both a time interval and any events within this interval. A process reacts to the receipt of an input message by updating its internal state; and, if outputs can be advanced in time,

The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

by sending messages to connected processes. These messages may include *null messages* that convey no events (changes in the state information), but serve only to advance the simulation time.

It is easy to show that such a simulator is correct [3], in the sense that it computes a possible behavior of the system being simulated. A sufficient condition for freedom from deadlock in this eager message-sending mode is that there is a positive delay in every circuit in the graph of element vertices and communication arcs. Intuitively, it is the delay of the elements being simulated that permits the element simulators to compute the outputs over an interval that is later than the time of the inputs, so that time advances. Simulation time is determined locally, and may get as far out of step at different elements as their causal relationships permit.

This conservative (also known as pessimistic) type of simulator exploits precisely the concurrency inherent in the system being simulated. In practice, just as with other concurrent programs, if the number of concurrently runnable processes substantially exceeds the number of processors, the utilization of concurrent resources is high. The speculative (also known as optimistic) type of simulator attempts to exploit additional concurrency by computing beyond the interval during which inputs are defined, at the risk of having to roll back if the speculations prove incorrect. Such approaches are attractive for simulating systems whose inherent concurrency is insufficient to keep concurrent resources busy, and in which speculations can be made with high confidence. Our studies have concentrated on conservative variants of the CMB algorithm.

The principal trouble with naive implementations of conservative CMB distributed simulation programs is a volume of null messages that may greatly exceed the number of event-containing messages. This difficultly is most evident when simulating systems with many short-delay circuits having relatively low levels of activity.

In practice, an element simulator may take as long to process a null message as an event-containing message, particularly with simple elements such as logic gates. In distributing the simulation, we seek to reduce the time required to complete the computation; however, we have an immediate problem if the element simulators must perform many more message-processing operations in the distributed simulation than they would perform event-processing operations in a sequential simulation. The centralized regulation of the advance of time achieved through the ordered event list maintained by sequential simulation programs allows these simulators to invoke element routines only once for each input event. The null messages inflate not only the volume of messages the system must handle, but also the computational load. Thus, if we are going to compete with the best sequential simulators, we must reduce the volume of null messages.

3. Indefinite Lazy Message Sending

To reduce the volume of messages, we use various strategies to defer sending outputs in the hope that the information can be packed into fewer messages. For example,

one of the most obvious schemes is to defer sending null messages, so that a series of null messages and an event-containing message can be combined to form a single message that spans a longer interval. Since output events are often triggered only by input events, deferring the delivery of preceeding null messages is less likely to hamper the progress of the destination element than deferring the delivery of event-containing messages.

The first problem that must be addressed in employing such strategies is deadlock. When element simulators defer sending output messages, they may cyclically deny themselves input messages, leading to deadlock. All of our simulators have employed a technique of *indefinite lazy message sending* to permit arbitrary strategies for deferring message sending, while still avoiding deadlock. The following is the inner loop of the simulator, shown in the C programming language:

```
while(1)
    if (p = xrecv())
        simulate_and_optionally_send_messages(p);
    else
        take_other_action();
```

The function `xrecv` returns a pointer, `p`, that points to a message for the simulation process if a message has been received. The simulator then dispatches to the appropriate element simulator, and may either send or queue the outputs that the element simulator produces. If there is no message in the node's receive queue, the pointer returned is a NULL (0) pointer. In this case, the simulator takes other action to break any possible deadlock. For a source-driven simulator, it selects a queued output to send as a message. For a demand-driven simulator, it selects a blocked element, and sends a *demand* message to its predecessor to request that queued outputs be sent. A deadlock in deferring messages cannot occur without "starving" a node of messages. When this situation is detected by `xrecv` returning a NULL pointer, the resulting action breaks the potential deadlock.

Within this indefinite lazy message-sending framework, we can experiment with *any* scheme for deferring and combining messages without concern for deadlock. A message is free to carry any number of events, and an element is free to defer message sending on any basis.

4. Variant Algorithms

We have experimented with many CMB variants; in the interests of comprehension, we will outline the operation and report the performance of six variants that are representative of the range of possibilities that we have studied:

A ***Eager message sending:*** This basic form of CMB serves as a baseline for comparison against the variants.

B ***Eager events, lazy null messages:*** Null outputs are queued. Event outputs are sent immediately combined with any queued null outputs. When `xrecv` returns

a NULL pointer, the null output that extends to the earliest time is sent as a null message.

C ***Indefinite lazy, single event:*** All output from element simulators is queued. Messages are sent only when `xrecv` returns a NULL pointer. The output queue that extends to the earliest time is selected to generate a message up to the first event, if any, or a null message to the end of the interval.

D ***Indefinite lazy, multiple event:*** This scheme is a slight variation on *C*, motivated by characteristics of multicomputer message systems that make it economical to pack multiple events into fewer messages. All output from element simulators is queued. The output queues may contain multiple events. When `xrecv` returns a NULL pointer, the output queue that extends to the earliest time is selected to generate a message up to the *last* queued event, if any, or a null message to the end of the interval. However, to allow a direct comparison with sequential simulators, events are processed singly.

E ***Demand driven:*** Although we usually think of simulation as source driven from inputs, one can equally well organize the simulation as demand driven from outputs. In the pure demand-driven form, all output from element simulators is queued. When `xsend` returns a NULL pointer, the input that lags furthest behind selects the destination for a demand message. Upon receipt of a demand message, if the output queue is not empty, the simulator sends all the information in the output queue; if the output queue is empty, the simulator generates another demand message to the source of lagging input to this element.

F ***Demand driven adaptive:*** Demand messages single out critical paths in a simulation. In an adaptive form of demand-driven simulation, a threshold is associated with each communication path. Outputs of element simulators are queued only up to the threshold; when the threshold is exceeded, the contents of the queue are sent as a message. Demand messages operate as in *E*, but also cause the threshold to be decreased (in the cases shown below, the threshold is halved). The simulator is accordingly able to adapt itself to the characteristics of the system being simulated.

Although these variants are described here in terms of message passing, the same variants also appear as different scheduling strategies in shared-memory implementations.

5. Experimental Method

In common with other highly evolved message-passing programs, the simulator is implemented with one simulation process per multicomputer node (or, in the Cosmic Environment, with one simulation process per host computer or per processor in a multiprocessor). The instrumented simulator is actually a simulator within a simulator.

Basis of comparison: Although real-time execution speed is one of the most natural bases of comparison between any two programs that perform the same

function, real-time speed and speedup curves are not themselves particularly revealing when there are so many parameters involved.

In order to unmask the behavioral differences of the simulators, we normalize the measured execution speeds to a common unit, called a *sweep* [5, 6]. Here we will let a sweep be a fixed time required to process one message, whether a single event, null message, or demand message. The number of sweeps required for a sequential simulator to complete a simulation is simply the number of events generated during the simulation.

Instrumentation: The simulator is a reactive program written in C, and is instrumented to function in two operational modes. In the *emulation mode*, a multicomputer emulation program runs a simulation of a multicomputer; this in turn runs the reactive simulators. Speed is measured in sweep units. On each sweep, each node is allowed to get one message from its receive queue (if not empty) and process it. In the *real mode*, the simulator runs directly on the multicomputer. There is one copy of the simulator process in each node, and each simulator process runs a subset of the elements as embedded reactive processes. Each node runs at its own pace, and speed is measured with UNIX's real-time clock.

6. Experimental Results

We have performed these studies using logic circuits, because it is easy to construct examples with a diversity of behaviors, and because logic simulation is itself of practical interest. Performance measurements have been made on a variety of logic circuits, including those that are representative of circuits found in computers and VLSI chips, and those that are designed specifically to test or to stress the simulator. Six different network types, each in several sizes up to 4000 logic gates, have been the principal vehicles for these experiments. A larger range in performance is observed among circuits with different characteristics than between algorithm variants.

Multiplier example: The parallel multiplier is a good example of an ordinary logic circuit. It contains only limited concurrency: An n-bit multiplier has an average concurrency of $2n$ due to the sequential dependency in the paths for carry and sum. It does not contain tight loops that give the simulator artificial boosts or troubles, depending on element distribution and loop stability. It also contains moderately high fanout in the multiplier and multiplicand lines, which puts pressure on the message system. In all fairness, the distributed simulation of this multiplier circuit is not expected to do too badly or too well on a multicomputer.

For the simulation, the most-significant bit of the product is connected back to the multiplier input via an inverting delay. The delay is such that the multiplier reaches a stable state before the multiplier input changes. The multiplicand input is set to a value that causes the circuit to oscillate. A trace of the product outputs shows that the simulator and the circuit are running correctly.

Measurements in the emulation mode: In the emulation mode, a 14-bit multiplier is used. Each full adder is composed of seven logic gates, and the 14×14 structure contains a total of 1376 logic gates. The average number of concurrent events

is about 28. The plot in Figure 1 portrays in a log-log format the sweep count versus the number of nodes, N. The heavy horizontal line represents the number of sweeps a sequential simulator requires. The first remarkable characteristic of these performance measures is that they are so similar across this class of variant algorithms.

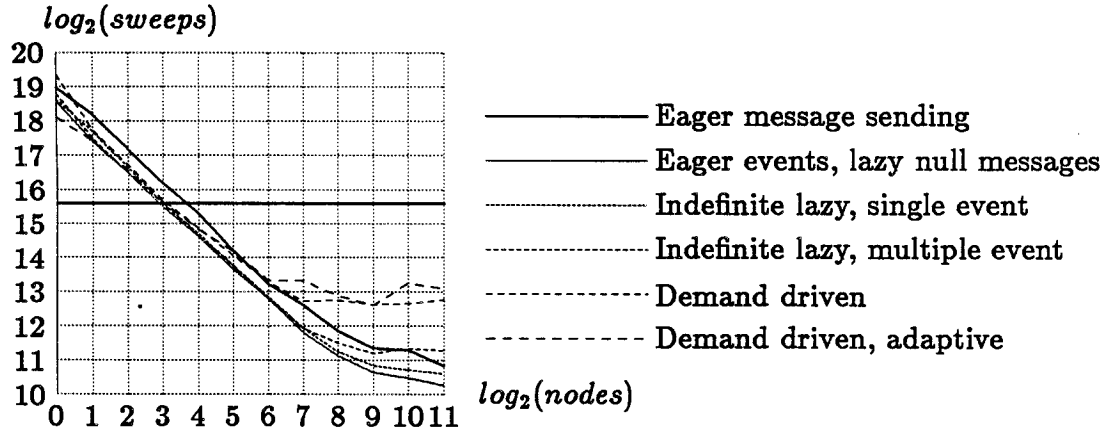

Fig 1: A 1376-gate multiplier, emulation mode

At $N=2^0=1$ node, we can compare the CMB variants with the sequential event-driven simulator. The concurrent simulators produce 4–10 times as many null or demand messages as event-containing messages, which is consistent with the 2–3 octave increase in sweep count over that of the sequential simulator. The speedup is close to linear in N for 5–8 octaves. The concurrent simulators do not become competitive with the sequential simulator until about $N=8$, but continue to nearly halve the sweep count with each doubling of resources until limiting effects are reached.

The demand-driven simulation modes E–F begin to perform poorly due to an increase in the volume of demand messages when the available concurrency of 28 ($\approx 2^5$) in the system being simulated is exhausted. In the adaptive form, demand messages are meant to make small-delay circuits more eager by reducing their queueing threshold. However, because the multiplier does not contain any small-delay circuits, demand messages drive the queueing threshold too low, and cause an excessive volume of null messages.

The source-driven variants extend the linear speedup for about 3 more octaves until the extra concurrency introduced by the null messages is also exhausted. These simulators reach asymptotic minimal time at 5 octaves below that of the sequential simulator, with only 3–6 elements per node. At this point the available concurrency is exhausted, and the number of elements per node is too small for the weak law of large numbers to assure load balance. The placement of elements in nodes for these trials is balanced but random.

Additional statistics have been collected to measure other effects. For example, when there are many circuit elements per node, the simulators are quite insensitive

to latency. When there are few elements per node, the performance begins to deterioriate as message latency is increased, particularly for the variants that perform well.

A second example for comparison: Figure 2 shows the sweep count versus N for a 3400-gate clock network. This asynchronous sequential circuit has many small-delay closed signal paths and a high activity level, resulting in an average event concurrency of 256.

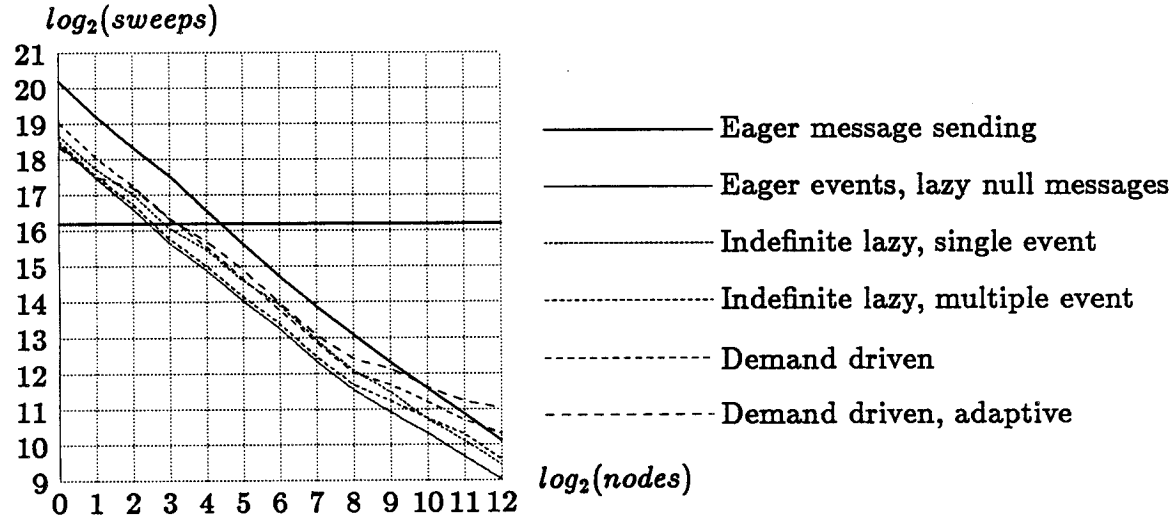

Fig 2: A 3400-gate clock network, emulation mode

Measurements on a real multicomputer: The results of simulating a scaled-down, 4-bit multiplier with 116 logic gates on an Intel iPSC/1 is shown in Figure 3. Simulation of larger circuits gives excellent but uninteresting results, with linear speedup over the entire range of $1 \leq N \leq 64$. (Due to limitations of the iPSC/1 message system, neither of the demand-driven simulation modes will run.) The timing results show that the reactive simulators require about twice as many calls to element simulators than a sequential simulator. The one-octave overhead is less than that of the 14-bit multiplier because a larger fraction of the elements are active. Since the average concurrency of the circuit is around eight, concurrency introduced by the circuit and by the null messages is expected to be exhausted when $N \geq 16$ nodes. Although the elapsed time plot shows that the time starts to level off when there are more than 16 nodes, it is somewhat less than linear in the range from 1–16 nodes, and is still decreasing slowly out to 64 nodes. The sublinear speedup is due to message latency in inter-node communications, increased null messages as the simulation is increasingly distributed, and load imbalance.

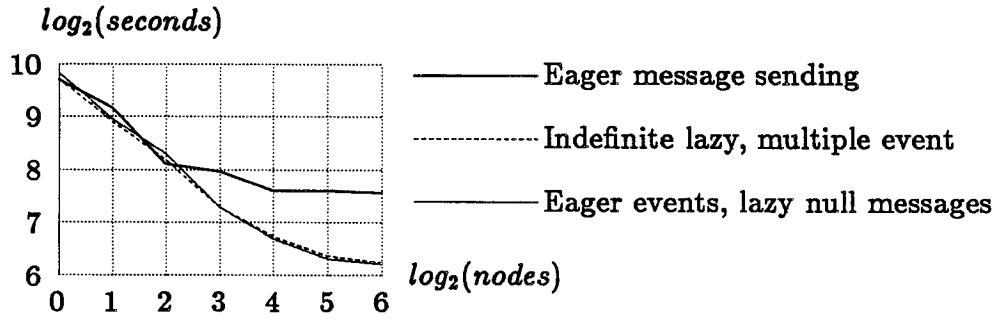

Fig 3: A 116-gate multiplier on an iPSC/1 for a 100μs period

7. Conclusions

Logic simulation, which involves simulating the behavior of relatively simple elements that have a high degree of connectivity, would be expected to be a difficult case for distributed simulation. Indeed, the simulations presented here have been much more revealing of the limitations of multicomputers and of the distributed discrete-event simulation algorithms than earlier simulations that we performed of systems such as multicomputer message networks.

For small N, neither the basic CMB algorithm nor the variants that we have tried are nearly as efficient for logic simulation as the sequential event-driven simulator. The null message is simply not as powerful a synchronization mechanism as the global ordered event list. However, for large logic circuits, these conservative variants on CMB produce excellent performance on multicomputers with large N and small message latency.

Our current efforts are to implement what we believe will be an entirely practical logic simulator for multicomputers and multiprocessors. It will employ a sequential event-driven simulator with an ordered event list in each node, and these simulators will be tied together using variants B, C, or D. Instead of random element placement, we will compute a placement that localizes small-delay circuits.

8. Acknowledgment

We very much appreciate the constructive suggestions, ideas, and encouragement that we have received from K. Mani Chandy.

9. References

[1] K. Mani Chandy and Jayadev Misra, "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations," *CACM* 24(4), pp 198–205, April 1981.

[2] Randal E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[3] Jayadev Misra, "Distributed Discrete-Event Simulation," *Computing Surveys* 18(1), pp 39–65, March 1986.

[4] "Submicron Systems Architecture," Semiannual reports to DARPA, Caltech Computer Science Technical Reports [5220:TR:86] and [5235:TR:86], 1986.

[5] William C. Athas and Charles L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer* 21(8), pp 9–24, August 1988.

[6] William C. Athas, "Fine Grain Concurrent Computation," Caltech Computer Science Technical Report (PhD thesis) [5242:TR:87], May 1987.

[7] William J. Dally, *A VLSI Architecture for Concurrent Data Structures,* Kluwer Academic Publishers, 1987.

[8] Charles L. Seitz, Jakov Seizovic, and Wen-King Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming," Caltech-CS-TR-88-1, January 1988.

Adaptive Routing in Multicomputer Networks

John Y. Ngai
Charles L. Seitz
California Institute of Technology*

Multicomputer Networks. Message-passing concurrent computers, more commonly known as *multicomputers*, such as the Caltech Cosmic Cube [1] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes [2]. The existing communication networks of the second-generation machines such as the Ametek 2010 employ an *oblivious* wormhole routing technique [6,7] which guarantees deadlock freedom. The message latency of these highly evolved oblivious technique have reached a limit of being as fast as physically possible while capable of delivering, under random traffic, a *stable* maximum substained throughput of $\approx$ 45 to 50% of the limit set by the network bisection bandwidth. Any further improvements on these networks will require an *adaptive* utilization of available network bandwidth to diffuse local congestions.

In an adaptive multi-path routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. It is expected that such an adaptive control can increase the throughput capability towards the bisection bandwidth limit, while maintaining a reasonable network latency. While the potential gain in throughput is at most only a factor of 2 under random traffic, the adaptive approach offers additional advantages such as the ability to diffuse local congestions in unbalanced traffic, and the potential to exploit inherent path redundancy in these richly connected networks to perform *fault-tolerant* routing. The rest of this paper consists of a brief outline of the various issues and results concerning the adaptive approach studied by the authors. A much more detailed exposition can be found in [3].

*The research described in this report was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597, and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

Adaptive Cut-through Routing. In any adaptive routing scheme which allows arbitrary multi-path routing, it is necessary to assure communication deadlock freedom. A very simple technique that is *independent* of network size and topology, is through voluntary *misrouting* as suggested in [4] for networks that employ data *exchange* operations, and more generally in store-and-forward networks. It was clear from the beginning that in order for the adaptive multi-path scheme to compete favorably with the existing oblivious wormhole technique, it must employ a switching technique akin to *virtual cut-through* [5]. In cut-through switching, and its blocking variant used in oblivious wormhole routing, a packet is forwarded immediately upon receiving enough header information to make a routing decision. The result is a dramatic reduction in the network latency over the conventional store-and-forward switching technique under light to moderate traffic. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly *matched*. A simple way is to have all *bidirectional* channels of the same node operate *coherently*. Observe that in the extreme, packets coming in can always be either forwarded or misrouted, even if the router has no internal buffer storage. In practice, buffers are needed to allow packets to be injected into the network, and to increase the performance of the adaptive control.

Network Progress Assurance. The adoption of voluntary misrouting renders communication deadlock a non-issue. However, misrouting also creates the burden to demonstrate progress in the form of message delivery assurance. An effective scheme that is independent of any particular network topology is to resolve channel access conflicts according to a *priority* assignment. A particularly simple priority scheme assigns higher priorities to packets that are closer to their destinations. Provided that each node has enough buffer storage, this priority assignment is sufficient to assure progress, *ie.*, delivery

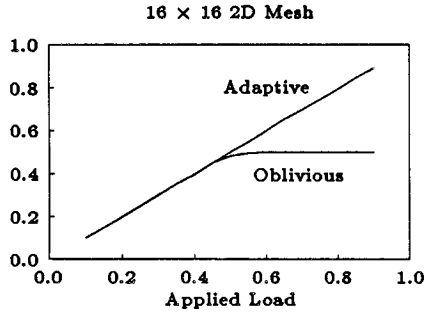

Figure 1: Throughput versus Applied Load.

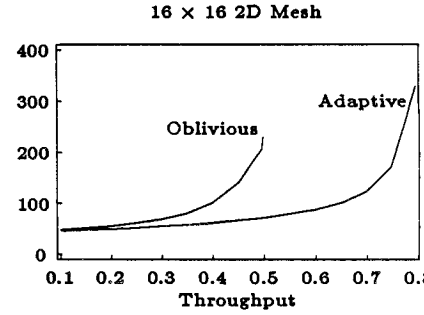

Figure 2: Message Latency versus Throughput.

of packets in the network. A more complex priority scheme that assures delivery of *every* packet can be obtained by augmenting the above simple scheme with *age* information, with higher priorities assigned to older packets. Empirical simulation results indicate that the simple distance assignment scheme is sufficient for almost all situations, except under extremely heavy applied load.

Fairness in Network Access. A different kind of progress assurance that requires demonstration under our adaptive formulation is the ability of a node to inject packets eventually. Because of the requirement to maintain strict balance of input and output data rates, a node located in the center of heavy traffic might be denied access to network indefinitely. One possible way to assure network access is to have each router set aside a fraction of its internal buffer storage exclusively for injection. Receivers of packets are then required to return the packets back to the senders, which in turn reclaim the private buffers enabling further injections. In essence, the private buffers act as *permits* to inject, which unfortunately have to be returned back to the original senders, thereby wasting network bandwidth. A different scheme that does not incur this overhead is to have the nodes maintain a bounded synchrony with neighbors on the total number of injections. Nodes that fall behind will, in effect, prohibit others from injecting until they catch up. With idle nodes handled appropriately, the imposed synchrony assures eventual network access at each node having packets queued for injection.

Performance Comparisons. An extensive set of simulations were conducted to obtain information concerning the potential gain in performance by switching from the oblivious wormhole to the adaptive cut-through technique. Among the various statistics collected, the two most important performance metrics in communication networks are *network throughput* and *message latency*. Figure 1 plots the substained normalized network throughput versus the normalized applied load of the oblivious and adaptive schemes for a 16 × 16 2D mesh network, under random traffic. The normalization is performed with respect to the network bisection bandwidth limit. Starting at very low applied load, the throughput curves of both schemes rise along a unit slope line. The oblivious wormhole curve levels off at $\approx$ 45 to 50% of normalized throughput but remains *stable* even under increasingly heavy applied load. In contrast, the adaptive cut-through curve keeps rising along the unit slope line until it is out of the range of collected data. It should be pointed out, however, that the increase in throughput obtained is also partly due to the extra silicon area invested in buffer storage, which makes available adaptive choices. Figure 2 plots the message latency versus normalized throughput for the same 2D mesh network for a typical message length of 32 flits. The curves shown are typical of latency curves obtained in virtual cut-through switching. Both curves start with latency values close to the ideal at very low throughput, and remain relatively flat until they hit their respective transition points, after which both rise rapidly. The transition points are $\approx$ 40% and 70%, respectively for the oblivious and adaptive schemes. In essence, the adaptive routing control increases the quantity of routing service, *ie.*, the network throughput, without sacrificing the quality of the provided service, *ie.*, the message latency, at the expense of requiring more silicon area.

Fault-tolerant Routing. Another area where adaptive multi-path routing holds promise is in fault-tolerant routing. The opportunity here stems from the fact that, as we continue to build larger machines, we expect faults to be increasingly probable. However, for performance reasons, the networks popular in multicomputers are already very rich in connectivity. It is conceivable that a multi-

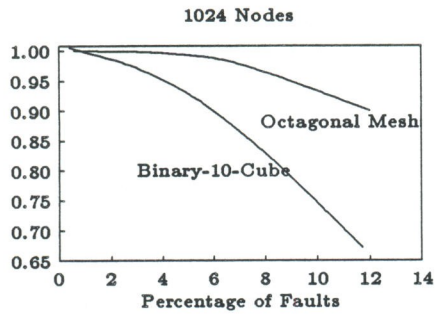

Figure 3: Reclamation Ratio for Node Faults

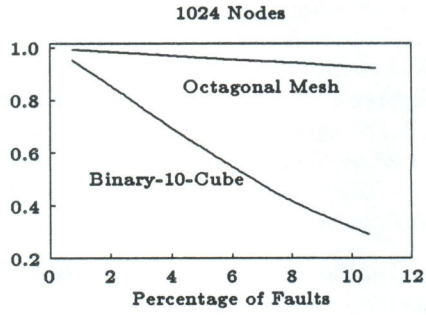

Figure 4: Reclamation Ratio for Edge Faults

path control can perform fault-tolerant routing simply by exploiting the *inherent path redundancy* in these networks. Fault-tolerant routing has been intensively studied in the network research community. However, multicomputer networks impose stringent restrictions, not present in traditional networks, that require a new approach. In particular, observe that the popular connection topologies of multicomputer networks such as k-ary n-cubes or meshs are highly *regular*, which allow for simple algorithmic routing procedures based entirely on local information. Such capability is particularly important in fine-grain multicomputers where resources at each node are scarce. Equally important, the simple algorithmic routing procedures in these regular topologies allow direct hardware realization of the routing functions, which is absolutely essential in high performance systems.

As nodes and channels fail, the regularity of these networks is destroyed and the algorithmic routing procedures are no longer applicable. Routing in irregular networks can be achieved by storing and consulting routing tables at each node of the network. However, such a scheme demands excessive resources at each node and becomes unacceptable as the networks grow in size. A different and more satisfactory approach exploits the regularity of the original non-faulty network. An interesting example of such an approach can be found in [8]. In this paper, we suggest an alternate approach based on our adaptive routing formulation. Instead of devising ways to route messages in these semi-irregular networks, we seek ways to *restore* the original regularity of the survival networks. This approach allows us to continue to use the original algorithmic routing procedure. One immediate advantage is that the faulty network can continue to use the original hardware router with very little change. Another advantage of this approach is that we can obtain *a priori* bounds on the length of routes joining pairs of sources and destinations in the faulty network.

Regularization Procedures. An immediate result of having only local information to guide routing is that, pairs of survived nodes may not be able to communicate with each other even if they remain connected. In order to communicate, each pair must have at least one unbroken *route* joining them, which belongs to the set of original routes generated algorithmically in the non-faulty network. Because of its resemblance to the notion of *convexity,* we refer to them as *convex networks*. Starting with an irregular survived network, one way to restore regularity is to selectively *discard* a subset of the survived nodes, so that the remaining subset becomes *convex*, and hence can still communicate with each other according to the original algorithmic procedure. In essence, nodes which become difficult to reach without global information are abandoned as a result of our insistence on using only local routing information. Another technique that can be employed to restore regularity is to selectively *restrain* a subset of the survived nodes to operate purely as routing switches, *ie.*, they are not allowed to source or consume messages. The rationale is that some survived nodes which are difficult to reach from everywhere, and hence should be discarded, may be in positions which enable other pairs to communicate, and hence should be retained.

Some Reclamation Results. It is clear that the effectiveness of this regularization approach will ultimately depend on the connection topology and the routing relations defined by the algorithmic routing procedure. High-dimensional networks such as the binary n-cube are expected to deliver good results, whereas low-dimensional ones such as the 2D meshes generally do not. One possible way to improve the reclamation yield of these low-dimensional networks is to *augment* them with extra channels, *eg.*, adding diagonal connected channels to a 2D mesh results in

an *octagonal mesh.* The additional connectivity in the octagonal mesh generates a much richer set of paths, and hence delivers much better reclamation yield. Figures 3 and 4 plot the reclamation ratio for the 32×32 octagonal mesh and Binary-10-cube versus the fraction of node faults, and channel faults respectively. The faults were generated independently and uniformly over the specific networks.

Future Challenge. Many aspects and problems have been addressed in the course of this research, and a number of solutions have been found. Clearly, more work remains to be done. Perhaps the most challenging of all is to realize on *silicon*, the set of ideas outlined in this study.

References.

[1] Charles L. Seitz, "The Cosmic Cube", *CACM*, 28(1), January 1985, pp. 22-33.

[2] William C. Athas, Charles L. Seitz., "Multicomputers: Message-Passing Concurrent Computers", *IEEE Computer*, August 1988, pp. 9-24.

[3] John Y. Ngai, *Adaptive Routing in Multicomputer Networks.* Ph.D. Thesis, Computer Science Department, Caltech. To be published.

[4] Borodin, A. and Hopcroft, J., "Routing, Merging, and Sorting on Parallel Models of Computation", *Journal of Computer and System Sciences*, **30**, pp. 130-145 (1985).

[5] P. Kermani and L. Kleinrock, "Virtual Cut-Through : A New Computer Communication Switching Technique", *Computer Networks* 3(4) pp. 267-286, Sept. 1979.

[6] William J. Dally and Charles L. Seitz, "The torus routing chip", *Distributed Computing*, 1986(1), pp. 187-196.

[7] Charles M. Flaig, *VLSI Mesh Routing Systems.* Caltech Computer Science Department Technical Report, 5241:TR:87.

[8] J. Hastad, T. Leighton, M. Newman, "Reconfiguring a Hypercube in the Presense of Faults." *Proceedings of the 19th Annual ACM Symposium on Theory of Computing.* May, 1987.