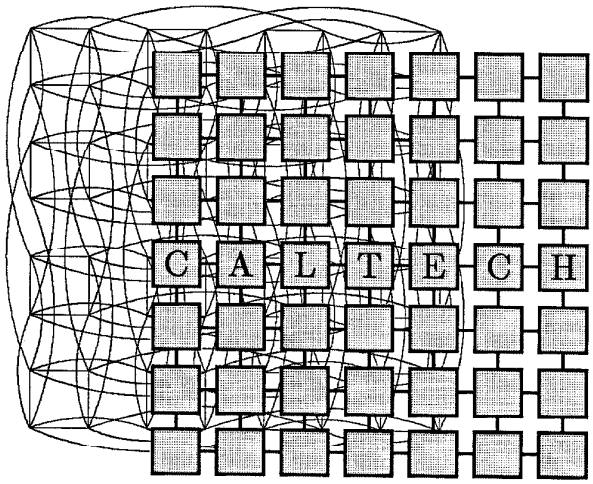

SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-89-4

31 March 1989

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.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
31 MAR 1989		**01-11-1988 to 31-03-1989**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Submicron Systems Architecture	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Defense Advanced Research Projects Agency,3701 North Fairfax Drive,Arlington,VA,22203-1714**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**18**	

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-89-4

31 March 1989

Reporting Period:	1 November 1988 – 31 March 1989
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	K. Mani Chandy
	Alain J. Martin
	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 research activities and results for the five-month period, 1 November 1988 to 31 March 1989, 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 Highlights

- Mosaic prototype approaching completion (2.1).
- Delivery of 2nd-generation multicomputers (2.2)
- Programming with composition (3.3)
- First asynchronous microprocessor (4.1).
- Fast self-timed mesh routing chips (4.2).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jordan Holt, Jakov Seizovic, Don Speck, Wen-King Su, Steve Taylor, Tony Wittry

The Mosaic C is an experimental fine-grain multicomputer, currently in development. Each Mosaic node is a single VLSI chip containing a 16-bit processor, a three-dimensional mesh router with each of its channels operating at 160Mb/s, a packet interface, at least 8KB of RAM, and a ROM that holds self-test and bootstrap code. These nodes are arrayed logically and physically in a three-dimensional mesh. We are working toward building a 16K-node ($32\times32\times16$) Mosaic prototype, together with the system software and programming tools required to develop application programs.

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small. The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system.

The Mosaic project includes many subtasks, which are listed below together with their current status:

Design, layout, and verification of the single-chip Mosaic node. The design and layout of the Mosaic C chip are now complete, and are going through extensive switch-level simulation tests, including the simulation of multiple nodes (see section 4.3). We expect to send a memoryless version of the node element to fabrication in about two weeks as a final check of the processor, packet interface, and router sections. These chips will be connected to external RAM and ROM to provide functional node elements for software development and host interfaces. Fabrication of the first chips in 1.2μm CMOS technology with RAM and ROM is anticipated in June 1989; quantity fabrication is anticipated in September 1989.

Internal self-test and bootstrap code. Since the Mosaic C is a programmable computing element, devoting a portion of the bootstrap ROM to self-testing greatly simplifies the logistics of producing these chips in significant quantity. The bootstrap and self-test code has been designed and is currently being written. The code will be tested using the ROM connected to the memoryless Mosaic C elements. Additional tests to the channels, which must be accomplished by the fabricator's automatic test equipment, are also being written.

Packaging. A preliminary packaging design based on TAB-packaged Mosaic

C chips was completed following a visit to Hewlett-Packard NID to understand their TAB packaging capabilities. The manufacturing and replacement unit contains eight nodes in a logical 2×2×2 submesh on a circuit-card module whose physical dimensions are approximately 2.5×5inches^2. These modules have stacking connectors that provide 160 pins on both the top and bottom, and are confined by pressure between motherboards to provide a three-dimensional connection structure that can be disassembled and reassembled for repair.

Cantor runtime system. A complete Cantor runtime system was written in Mosaic assembly code, and is now running correctly with a suite of small test programs under a Mosaic simulator on our medium-grain multicomputers (see section 3.1). This system provides the low-level implementation of message and process-creation primitives, and normally will be loaded as part of the Mosaic system initialization. The evolution of the Cantor programming language and the experience gained by use are two factors that are expected to affect continuing refinements to this system.

Cantor language, compiler, and application studies. A definition of a version of Cantor (3.0) with functions and limited message discretion was proposed in January 1989 by William C. Athas of UT Austin. We have been studying the changes in the runtime support that will be required by these improvements. In the interim, the definition and compiler implementation of Cantor 2.2 remain in use for application development.

Host interfaces and displays. The three-dimensional mesh structure of the Mosaic allows a very large bandwidth around the mesh edges. In order to initiate and interact with computations within the Mosaic, we must provide interfaces between the Mosaic message network and conventional computers and networks. One approach being studied is to use a memoryless Mosaic with a two-ported external memory as a convenient interface to workstation computers. Another external connection that is desired is a display interface. An elegant method that uses one 32×32 plane of a Mosaic as a rendering engine, frame buffer, and output video-conversion system has been developed. The detailed design of the video output generator that attaches to one edge of this 32×32 plane is now under way.

2.2 Second-Generation Medium-Grain Multicomputers*

Chuck Seitz, Joe Bechenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su

A 16-node Intel iPSC/2 was delivered in November 1988, and a 16-node Symult Series 2010, a second-generation medium-grain multicomputer developed as a

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

joint project between our research project and Symult Systems, Inc. (formerly Ametek Computer Research Division), was delivered in December 1988. Both of these systems have been used extensively for programming system developments, applications, and benchmarks. We have encountered very few system problems in running existing Cosmic-C application programs on either the Symult Series 2010 or Intel iPSC/2.

Application programs typical of those that were written for first-generation multicomputers run 8–10 times faster per node on the Symult Series 2010 and on the Intel iPSC/2 than on first-generation machines, such as the Intel iPSC/1. Applications involving latency-sensitive non-local message traffic exhibit more dramatic improvements, particularly on the Series 2010, due to cut-through message routing being included in the hardware of these second-generation multicomputers.

Delivery of a 64-node Series 2010 is expected on 31 March 1989, and our 16-node Series 2010 will be returned briefly to Symult to be upgraded to 32 nodes and retrofitted with some hardware improvements to the mesh termination and host interfaces. The 32-node Series 2010 will continue as our principal programming-system-development machine. The 64-node Series 2010 and the 16-node iPSC/2 will be made available to outside users through the Caltech Concurrent Supercomputing Facilities. Outside users will include researchers at Caltech, as well as those associated with the Rice-Caltech-Argonne-Los Alamos (NSF Science and Technology) Center for Research in Parallel Computation. These systems will also be available for use by researchers in the DARPA community; DARPA researchers should contact Chuck Seitz (`chuck@vlsi.caltech.edu`) to make arrangements for access.

We expect to expand both the Intel iPSC/2 and Symult Series 2010 to larger configurations by the early part of CY90.

Copies of the Cosmic Environment system have been distributed to 13 additional sites during this period, bringing the total copies distributed directly from the project to over 160.

An effort has been started to implement major extensions of the Cosmic Environment host runtime system and the Reactive Kernel node operating system. The new CE will be based internally on reactive programming, and will allow a more distributed management of a set of network-connected multicomputers. The extended RK will support global operations across sets of cohort processes, including barrier synchronization, sum, min, max, parallel prefix, and rank. Another extension will be the support of distributed data structures, such as sets and ordered sets. These new features will be implemented at the RK handler level, where the message latency is only a fraction of that at the protected user level. The implementation of these algorithms at the handler level permits the performance of global and distributed-data-structure operations in times that do not greatly exceed those of user-level operations dealing with single messages.

Our Caltech project continues to work with both Intel and Symult on the architectural design, message-routing methods and chips, and system software for medium-grain multicomputers. We expect to see additional major advances in the performance and programmability of these systems over the next two years. In 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.

2.3 Cosmic Cube Project

Wen-King Su, Jakov Seizovic, Chuck Seitz

The Cosmic Cubes that were built in our project in 1983 and the Intel iPSC/1 d7 that was contributed to the project in 1985 continue to operate very reliably. Overall usage has decreased somewhat with the appearance of the second-generation multicomputers, but the iPSC/1 continues to be used fairly heavily within the research group for discrete event simulations, and by Caltech students and faculty in Aeronautics for supersonic-flow computations.

Neither the 64-node or 8-node Cosmic Cubes exhibited any hard failures in this five-month period. The two original Cosmic Cubes have now logged 3.8 million node-hours with only four hard failures, three of which were chip failures in nodes, and one a power-supply failure. A node MTBF in excess of 1,000,000 hours is probable based on this reliability experience.

3. Concurrent Computation

3.1 Cantor

Nanette J. Boden, Chuck Seitz

Programming Fine-Grain Multicomputers

The experiments we reported previously in application programming using Cantor 2.0 and 2.2 have suggested a series of changes to the Cantor language. William C. Athas, who led the development of Cantor while he was a graduate student and post-doc in the project, and who is now at UT Austin, has incorporated these ideas into the definition of a new version of Cantor (3.0). The principal structural changes are the introduction of limited discretion in receiving messages according to type, and in the approach to implementing functions.

In developing the Cantor programming system for the Mosaic, we mean to allow for these changes so that we may change to Cantor 3.0 as soon as a new compiler is produced.

Cantor for the Mosaic

Development of Cantor runtime support for the Mosaic multicomputer has progressed significantly during the last five months. Initially, we defined a Cantor Abstract Machine (CAM) that represents an idealized machine for executing Cantor code. The CAM instruction set includes single instructions that encapsulate complicated Cantor operations, such as process creation and message passing. By design, the implementation of these operations can be varied within native code generators for experimenting with different strategies. With the Mosaic, for example, we use a macro-assembler that translates the implementation for each CAM instruction into Mosaic instructions.

The definition of the first version of the Cantor runtime system for the Mosaic consisted chiefly of freezing efficient implementations for process creation and message passing, and expressing them with Mosaic instructions. In the case of process creation, a software cache of available reference values is maintained on each node so that processes can be created with low latency. These reference values are later bound to actual processes by special *creator* processes located on each node that allocate memory for new processes. Receiving a message on the Mosaic is implemented by having the runtime system determine the destination process, and then run that process to absorb the message. The runtime system also communicates with the runtime systems on other nodes to manage resources within the node, eg, sending requests for more reference values to fill the software cache.

To evaluate different runtime system prototypes, we developed a Mosaic simulator that runs on existing medium-grain multicomputers, including the Cosmic

Cubes, Intel iPSCs, and the Symult 2010. A host program distributes the Mosaic code for a Cantor program to each simulated Mosaic node, and initiates computation by instantiating the *main* process of the Cantor program. Program output is achieved by instantiating a *console* process and passing its reference in messages.

Currently, our simulator is working on a test suite of simple Cantor programs. In the future, we plan to incorporate some of the more recent Cantor innovations, eg, functions and limited message discretion, into the simulator and into the runtime system. We are also planning experiments to evaluate different strategies for code distribution and memory allocation throughout Mosaic nodes.

3.2 Concurrent Logic Programming

Stephen Taylor

A commercially supported concurrent logic programming system was ported to our Symult Series 2010 multicomputer, and is available for all users of our project's multicomputers.

This system is composed of a compiler for the language *Strand*, and an environment for program development. The language provides an abstract message-passing framework for use in a variety of symbolic and system integration tasks. The system is also operational on Intel iPSC systems, networks of Suns, Mecho Transputer surfaces, PC Plug-in Transputer cards, Encore/Sequent shared memory machines, BBN Butterfly, and Atari personal machines. The system was used for a graduate course in compiler techniques this quarter, and will be used in a graduate course on concurrent programming in this coming quarter. It is also being used to study various applications in the *composition* research described in the following section of this report. Finally, a textbook describing the ideas embodied in the Strand system was recently completed, and will be published by Prentice-Hall in July 1989.

3.3 Programming with Composition

Mani Chandy, Stephen Taylor

We are interested in developing a notation for specifying concurrent algorithms and programs. Our goals are to support formal reasoning about program correctness and to provide efficient implementations of symbolic, numeric, and operating system codes. We have chosen *program composition* as a central notion due to its prevalence in both semantic models and program design methodologies.

During the past six months, we have considered the basic components of such a notation. Our conclusion is that there are four composition operators of importance. These operators are defined on program units; the method by which these units are implemented is relatively unimportant. It is natural to expect the notation to allow existing codes (written in Fortran, C, Lisp, Ada, *etc*) to be reused on

multicomputers. Moreover, the composition of these units will have a formal semantic characterization. To explore the utility of the notation, we are currently focussing on the hand compilation of non-trivial application codes. If performance results indicate that the notation is sufficiently efficient, we plan to build a compiler targeted to multicomputer architectures.

In the area of numeric computing we are studying a large fluid-flow problem developed in the department of Applied Mathematics at Caltech. This Fortran application computes the transition from a two-dimensional Taylor Vortex to three-dimensional wavy-vortex flow. Central to the application is a relaxation algorithm that employs a multigrid method. After benchmarking, we discovered that more than 70% of the execution time for the application was spent in the relaxation algorithm; thus, we decided to focus on this algorithm. Unfortunately, we arrived at a somewhat negative conclusion: The original algorithm was based on a sequential line-iteration scheme that afforded no opportunity for concurrent execution. As a result, we have converted the original code to use a point Gaussian relaxation algorithm; this appears more suitable. We are currently in the process of debugging a concurrent formulation of the algorithm.

In the area of symbolic computing we are studying a large automated reasoning program in conjunction with the Aerospace Corporation in Los Angeles. This program has been used extensively for checking the correctness of hardware specifications and Ada programs. A central component of the program is a *congruence closure* algorithm used for maintaining equality assertions. We began this research by investigating the opportunities for executing portions of this algorithm concurrently. This, again, led us to a somewhat negative conclusion: The granularity of typical invocations of the algorithm is too low to benefit from concurrent execution. We are now investigating a new algorithm that overlaps the execution of multiple equality assertions. Since a large number of these occur in a typical proof, we believe this to be a more suitable direction.

Finally, we are also interested in working with DNA sequencing programs, but have not yet made substantial progress in this area.

It should be understood that the objective of these application efforts is to test the utility of the program-composition notation, rather than to develop the applications themselves.

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

Wen-King Su, Chuck Seitz

During the past five months, additional simulations using the new logic simulator have been made, and a revision of the paper "Variants of the Chandy-Misra-Bryant Distributed Discrete-event Simulation Algorithm" (included as an appendix to this report) was written for publication in the 1989 SCS Eastern Multi-Conference. A

test version of the hybrid simulator has been implemented on top of the concurrent CMB variant simulators. Results from this prelimiary investigation are promising, and a new, more efficient version of the hybrid simulator is currently being written.

3.5 Distributed Snapshots

Mani Chandy

One of the fundamental problems in distributed systems appears trivial: Record the state of the system. The problem is, however, quite difficult because distributed systems do not have a single system-wide clock. If there were a clock, all processes could record their local states at a predetermined time. The problem of recording global states of distributed systems is at the core of a large number of problems in distributed systems, including deadlock detection, termination detection, and resource management. The paper, "The Essence of Distributed Snapshots," submitted to the *ACM Transactions on Computer Systems*, and included as an appendix to this report, presents necessary and sufficient conditions for a collection of local snapshots (recordings of local states) to be a global snapshot. The paper shows that many distributed algorithms can be developed in a systematic and straightforward manner from these conditions.

4. VLSI Design

4.1 The Design of the First Asynchronous Microprocessor

Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, Pieter J. Hazewindus

We have completed the design of an entirely asynchronous (self-timed, delay-insensitive) microprocessor. It is a 16-bit, RISC-like architecture with independent instruction and data memories. It has 16 registers, 4 buses, an ALU, and two adders. The size is about 20,000 transistors. Two versions have been fabricated: one in 2μm MOSIS SCMOS, and one in 1.6μm MOSIS SCMOS. (On the 2μm version, only 12 registers were implemented in order to fit the chip into the 84-pin $6600\mu m \times 4600\mu m$ pad frame.)

With the exception of *isochronic forks* (see the paper included as an appendix to this report), the chips are entirely delay-insensitive, *ie*, their correct operation is independent of any assumption on delays in operators and wires except that the delays be finite. The circuits use neither clocks nor knowledge about delays.

The only exception to the design method is the interface with the memories. In the absence of available memories with self-timed interfaces, we have simulated the completion signal from the memories with an external delay. For testing purposes, the delay on the instruction memory interface is variable.

In spite of the presence of several floating n-wells, the 2μm version runs at 12 MIPS. The 1.6μm version runs at 18 MIPS. (Those performance figures are based on measurements from sequences of ALU instructions without carry. They do not take advantage of the overlap between ALU and memory instructions.) Those performance results are quite encouraging given that the design is very conservative: It uses static gates, dual-rail encoding of data, completion trees, *etc.*

Only two of the 12 2μm chips passed all tests, but 34 out of the 50 1.6μm chips were found to be entirely functional. However, within a certain range of values for the instruction memory delay, the 1.6μm version is not entirely functional. We cannot yet explain this phenomenon.

We have tested the chips under a wide range of VDD voltage values. At room temperature, the 2μm version is functional in a voltage range from 7V down to 0.35V! And it reaches 15 MIPS at 7V. We have also tested the chips cooled in liquid nitrogen. The 2μm version reaches 20 MIPS at 5V and 30 MIPS at 12V. The 1.6μm version reaches 30 MIPS at 5V. Of course, these measurements are made without adjusting any clocks (there are none), but simply by connecting the processor to a memory containing a test program and observing the rate of instruction execution. The results are summarized in Figure 1. The power consumption is 145mW at 5V, and 6.7mW at 2V. Figure 2 shows that the optimal *power-delay product* is obtained at 2V at room temperature.

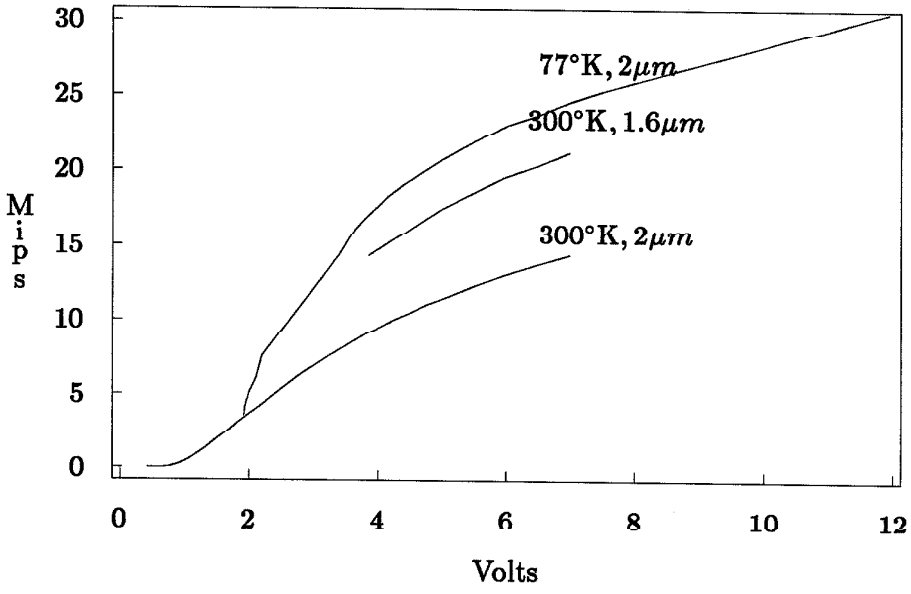

Figure 1: MIPS as a function of VDD

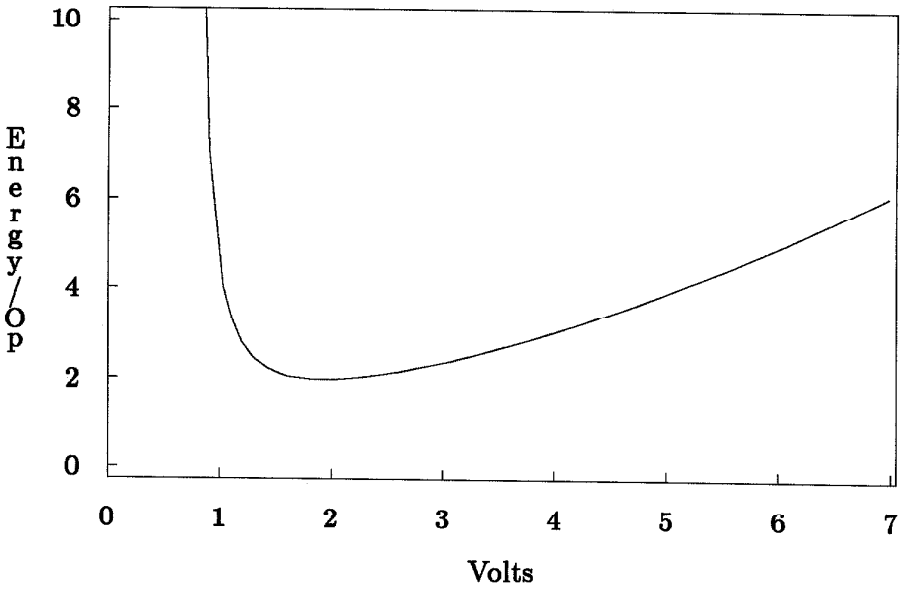

Figure 2: Power-delay product as a function of VDD

4.2 Fast Self-Timed Mesh Routing Chips

Chuck Seitz

The latest mesh-routing-chip (MRC) design, the FMRC2.1 design, was sent to MOSIS for 1.6μm SCMOS fabrication on 7 November 1988. This chip is a revision of FMRC2.0 that corrects a timing error in the latching of a routing decision. A Spice simulation indicated that that the revision corrected a timing error of approximately 0.7ns to a timing margin of about 1.0ns (about 50% of the difference between two short delay paths; hence, not as risky as it may sound). The maximal throughput predicted both by Spice and by tau-model calculations was 60MB/s.

These chips were returned from fabrication on 10 January 1989, and were found to operate correctly under a nearly exhaustive functional screening, and at a maximum throughput of 56MB/s. The yield on this run was 44/50. One of the chips had a cracked package, and two had bonding shorts; hence, the fabrication yield was actually 44/47.

Batches of 20 good chips were sent both to Intel Scientific Computers (as GFE on their DARPA contract) and to Symult Systems, and both companies have verified that these chips operate correctly in their test fixtures or systems.

The FMRC2.1 chip employs a design method that is *not* entirely delay-insensitive (see previous section). The circuit exhibits races *within* modules, but these modules have self-timed interfaces to other modules. Previous MRCs, entirely pin-for-pin compatible, employed the same delay-insensitive style as the asynchronous processor reported in the previous section, and required nearly twice the silicon area to operate half as fast as the FMRC2.1.

Hence, we conjecture that we shall see the same phenomenon with self-timed designs that is apparent with conventional designs; namely, that chips with relatively few cell types, such as memories and MRCs, will profitably employ circuit-level optimizations. Such optimizations are relatively less profitable and manageable in more complex chip designs, such as processors.

4.3 Mosaic C Chip

Jakov Seizovic, Jordan Holt, Chuck Seitz, Don Speck, Wen-King Su, Tony Wittry

During the past few months, work on the Mosaic chip has predominantly consisted of a series of extensive switch-level simulations. Using COSMOS instead of MOSSIM, we were able to decrease the simulation time by a factor of ten, with a negligible additional cost in setup (compile) time. The simulation of a memoryless version of Mosaic chip, consisting of about 26K transistors, takes slightly over a second of real time per clock cycle when running on a SUN 3/260. This has enabled us to simulate fairly long sequences of instructions from the Cantor runtime system at the switch-simulation level.

Having completed simulations of all of the logic parts of the Mosaic chip, *ie*, processor, packet interface, router, and bus arbiter, independently as well as together, we are entering the final phase of switch-level simulations, where multiple Mosaic chips will be represented as processes under CE/RK, and run on the multicomputers operated by the project, as well as on workstations.

We are planning to send the first version of a Mosaic chip to fabrication on a 2μ MOSIS run within a couple of weeks.

4.4 New CMOS PLAs

Jakov Seizovic, Chuck Seitz

A NOR-NOR precharged PLA has been designed to replace the NAND-NOR precharged PLA that we have used extensively since 1985. Both the delay and precharge time of this NOR-NOR PLA are linear in the number of inputs, a significant improvement compared to the NAND-NOR PLA, in which the delay is quadratic, and precharge time is cubic. This PLA has replaced the two NAND-NOR PLAs in the Mosaic C packet interface and the hybrid static/precharge NAND-NOR PLA in the Mosaic processor, and accordingly has saved us a lot of time and trouble in the Mosaic design.

4.5 CIF-flogger

Glenn Lewis, Chuck Seitz

CIF-flogger is a multicomputer program for flattening CIF files, rasterizing the geometry, and performing parallel operations on the geometry in strips. It runs under the CE/RK system, and hence, on most available multicomputers, including the Intel iPSC/2 and Symult Series 2010.

CIF-flogger currently supports the following operations on the chip geometry:

- parsing the CIF specification file (produced by Magic)
- flattening and rasterizing the hierarchical design geometry
- recognizing transistor geometry
- global connected-component labeling
- bloat, shrink, and logical mask layer operations
- creating new CIF for a processed design

Plans for CIF-flogger include:

- general CIF-reading capability
- circuit extraction

- well-plug checking
- design-rule checking

Initial timings indicate that CIF-flogger provides these operations in a matter of a few seconds for 100K-transistor chips. CIF-flogger is intended to be a useful tool for chip designers and foundries to verify that a design passes "syntactical" checks before it is fabricated, thus saving both time and money.

4.6 Adaptive Routing in Multicomputer Networks

John Y. Ngai, Chuck Seitz

As we are wrapping up our theoretical investigation of multicomputer adaptive routing, our recent efforts have been concentrated in two areas:

(1) The first of a series of publications will appear in the 1989 ACM Symposium on Parallel Algorithms and Architectures, to be held in Sante Fe, New Mexico this June. (A copy of this paper is included at the end of the report.)

(2) We have been searching for practical implementation ideas for replacing the existing oblivious router in the Mosaic with an adaptive router. A low-latency header encoding and modification scheme that we have dubbed the "sign-first one-shy code" has been devised for an adaptive router with a relatively narrow flit width. The details of these implementation ideas can be found in a forthcoming PhD thesis.