SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-90-05

15 March 1990

Reporting Period:	1 November 1989 – 15 March 1990
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 four-and-one-half-month period, 1 November 1989 to 15 March 1990, 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 is ready to build (section 2.1).
- Fully functional Memoryless Mosaic chips (section 2.1.4).
- High-density Mosaic memory (sections 2.1.2 and 4.7).
- Mosaic program-development boards (section 2.1.5).
- New message-order semantics (section 3.2).
- Cache memory for an asynchronous microprocessor (section 4.2).
- New results in transistor-sizing for asynchronous circuits (section 4.4).

2. Architecture Experiments

2.1 Mosaic Project

Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su

The development of the Mosaic C, an experimental *fine-grain multicomputer* based on single-chip nodes and a reactive-process programming model, is entering its final stages. This system-building experiment incorporates much of what we have learned over the past decade about the architecture, design, and programming of multicomputers. Indeed, many of our recent contributions to the development of medium-grain multicomputers (see section 2.2), such as low-latency message-passing networks and streamlined message handling in the node operating system, have come directly out of our investigations of the design and programming of fine-grain multicomputers, in which these problems are substantially more difficult.

The Mosaic C project includes numerous interacting subtasks ranging from chip design and system packaging to programming-system development and application studies. The fabrication of a large-scale prototype is now forcing decisions on design options that have deliberately been left open; hence, we offer in this semi-annual technical report a detailed status report on the entire project.

2.1.1 Architecture rationale

The Mosaic C is a member of a class of programmable, MIMD, distributed-memory, concurrent computers called multicomputers. (See the article by Athas & Seitz in the August 1988 issue of *IEEE Computer* for background.) These machines consist of an ensemble of N programmable computers called *nodes,* each of which may support many concurrent processes. Interprocess communication takes place by messages that are conveyed and routed between nodes by a direct communication network. Multicomputers are true VLSI architectures: They can be scaled to very large numbers of nodes, and can exploit the performance and complexity of submicron-feature-size microelectronic technologies. Multicomputers have proven to possess a broad application span, and allow explicitly concurrent programs to be expressed in a variety of programming notations.

The commercial examples of multicomputers manufactured by Intel Scientific Computers, Symult Systems, and N-CUBE are based on a computational model, prototype developments, and system software developed in our research project. They are all *medium-grain multicomputers* in which configurations capable of substantially outperforming conventional vector supercomputers consist of hundreds of nodes with several MBytes of storage per node.

Shared-memory multiprocessors are not as scalable as multicomputers; however, multiprocessors can certainly be scaled into the range of hundreds of processors, and in this range possess some advantages over multicomputers. Among MIMD systems, the exclusive "niche" of the multicomputer begins at about $N \geq 2^{10}$ nodes. We understand today how to scale multicomputers to at least $N = 2^{21}$ nodes.

Although medium-grain machines can be scaled into the range of thousands of nodes, economics dictates that multicomputers with large N will employ small nodes. Consider this constant-silicon-cost argument. A medium-grain multicomputer with $N = 256$ and 4MB/node requires about 1m^2 of silicon in a modern 1μm CMOS process. About 60% of the $4{,}000\text{mm}^2$ silicon area of each node is devoted to the 4MB of primary memory. Suppose that the essential parameters of a multicomputer design, N and the node size, were shifted by a factor of 2^6, so that a machine would consist of 16K nodes, each with 64KB of memory. Such a machine would have the same total memory and silicon-area cost as a 256-node medium-grain multicomputer; however, because the performance of the instruction-interpreting processor is not reduced in proportion to its area, the aggregate peak performance of this fine-grain multicomputer system would be significantly higher than that of a medium-grain multicomputer. In fact, because a single node would require only about 60mm^2 and could be integrated onto a single chip, the localization of communication between the processor and memory allows a single-chip node to exhibit performance that is comparable to that of the multi-chip node used in medium-grain systems.

The Mosaic C closely fits this description of a fine-grain multicomputer. It is based on single-chip nodes, and we are working toward assembling a prototype consisting of 16K nodes. We recognized long ago that multicomputers with single-chip nodes were technologically the most attractive point within the space of multicomputer designs. As was reported in 1985 (see Seitz's article in the January 1985 issue of the *CACM*), the Cosmic Cube was developed by our research group (in 1981–83) to study the programming techniques and applications of the multicomputer systems that we expected could be constructed with single-chip nodes by about 1991.

We expect that the Mosaic C will become the origin of a new scaling track for multicomputers. The fine-grain, single-chip-node track offers substantially higher performance and performance/cost than medium-grain multicomputers, and is centered in a niche that is beyond the scaling range of multiprocessors, while still providing the wide application span of MIMD systems.

2.1.2 The Mosaic C node

Because single-chip nodes were a stipulation of the Mosaic experiment, it is most convenient to describe this system "bottom-up," starting from the single-chip node element.

The Mosaic C node was designed and laid out using the MOSIS SCMOS scalable-CMOS design rules, and uses fully restored logic with two-phase clocking. It is typical of chips designed with these rules and disciplines to be highly tolerant of process variations. The 50C design clock rate is 40MHz at 4V in 1.2μm SCMOS, and tests of parts fabricated in 1.6μm CMOS confirm that we will achieve this performance by a considerable margin.

The major parts were initially fabricated separately for testing and yield characterization, and are listed below:

Part	Lambda dimensions	As fabricated in 1.2um CMOS
16KB 4T dRAM	14000, 7700	8.4mm x 4.6mm = 38.6 sq mm
64KB 1T dRAM	14000, 12000	8.4mm x 7.2mm = 60.5 sq mm
8KB bootstrap ROM	7000, 3000	4.2mm x 1.8mm = 7.6 sq mm
Processor	4000, 3000	2.3mm x 1.8mm = 4.3 sq mm
Router	1500, 3000	0.9mm x 1.8mm = 1.6 sq mm
Packet Interface	1500, 3000	0.9mm x 1.8mm = 1.6 sq mm
TOTAL (16KB dRAM)	14000, 10700	8.4mm x 6.4mm = 53.8 sq mm
TOTAL (64KB dRAM)	14000, 16000	8.4mm x 9.6mm = 80.6 sq mm

These dimensions are slightly exaggerated to allow for the routing space between the parts. Allowing also for the pad frame and space to route signals to it, the chip dimensions for the version that uses the 16KB 4T dRAM will be approximately 9.0mm$\times$7.4mm = 67mm^2, and for the version that uses the 64KB 1T dRAM will be approximately 9.0mm$\times$10mm = 90 mm^2. The average power consumption for either design will be about 0.5W.

Because the *memory* uses the largest area and is the most difficult part of the design, two alternative memory designs were developed. The 16KB 4T dRAM is a conservative 4-transistor dynamic RAM designed as a low-risk option in case a higher density dRAM proved to be infeasible. This 4T dRAM is based on a cross-coupled n-channel cell. Data bits are in double-rail form, and reading is accomplished by precharging both data lines and then applying the word select. Writing is accomplished by driving the data lines to complementary values and applying the word select. The RAM performs a memory cycle on every clock cycle. In 1.2μm CMOS, it has an access time less than 20ns, and a cycle time of 25ns. The 64KB 1T dRAM is an aggressive, one-transistor-per-bit design that was completed in January 1990, and will be submitted for first full-scale fabrication on the MOSIS 1.2μm SCMOS run that is closing on 20 March 1990. (Several test structures have been fabricated and tested to verify the operation of circuits used in this dRAM.) The design of the dRAM is described in detail in section 4.7.

The *bootstrap ROM* is single-transistor mask programmable, and its read-cycle timing and organization is identical to that of the dRAM. The size listed, corresponding to 4K words, is much larger than necessary. The self-test, initialization, and bootstrap functions require approximately 600 words. However, because ROM is denser than RAM, it may be useful in future systems to put standard subroutines (such as for floating-point arithmetic) in the ROM so as to save space in the RAM.

The 16-bit, microcode-driven *processor* is the only source of addresses in the node, and performs a memory cycle on every clock cycle. The processor datapath includes 24 general registers and 12 addressing and special registers. The instruction set is similar to that of other RISC processors, with 8 addressing modes for the move instructions, ALU operations including integer multiply, conditional branch instructions, a subroutine call, and control instructions. Projected performance using our present compilers and clock-by-clock microprogram simulation is 14 MIPS (16-bit operands).

The unusual features of the Mosaic processor are motivated by its use in a multicomputer node. The refresh and packet-interface address control are actually part of the processor, and the processor microcode interleaves instruction execution from four sources: two program contexts, refresh operations, and transfer between memory and the packet interface. The processor's address registers include two program counters, one for user code and the other for message-system control, with zero-time context switching between them. The two pointers and two limit registers for the send and receive queues are also in the address register set, together with the refresh address register. The remaining special registers control the interrupt status of the packet interface and the dx, dy, dz values in the header of messages that are being sent.

Either of two *routers* can be used. The 3D synchronous router consists of three cascaded 1D routing automata with a 4-bit-data path. A unidirectional external channel is 6 wires, consisting of 4 data lines, one escape bit for control codes, and the reverse flow-control signal. Bidirectional channels in each of 6 directions for 3D routing thus require a total of 72 external pins. The bandwidth per channel is one 4-bit data item each clock period, or 20MB/s. The 2D asynchronous router consists of two cascaded 1D routing automata with an 8-bit-data path. It is a variant on the FMRC2 routers developed for medium-grain multicomputers. A unidirectional external channel consists of 8 data lines, tail bit, request, and acknowledge. Bidirectional channels in each of 4 directions for 2D routing require 88 external pins. The bandwidth per channel in the 1.2μm CMOS technology will be approximately 80MB/s.

The *packet interface* includes 4 words of FIFO in each direction, the 16-bit-to-4/8-bit and 4/8-bit-to-16-bit conversion logic, and the logic that generates the message header on sending. The arbiter for deciding whether the system should perform memory refresh, channel data accesses, or processor access is also in the packet interface; the decisions that it generates are inputs to the processor microcode. The refresh signal is an input to the chip, and is bused through an entire array of Mosaic elements. The reason for synchronizing the refresh operation is that packets that are bound for a node that is refreshing would otherwise be blocked into the message network, and block other messages that are in transit. Thus, one might as well refresh all of the nodes at once.

The Mosaic parts are quite modular, and can be assembled in a variety of floorplans. The principal internal interface is the memory bus, which consists of 16 data lines, 16 address lines, the write signal, and the clock and reset. In addition, there are several signals between the processor and packet interface, and two channels between the packet interface and the router.

2.1.3 Choice of network dimension

A Mosaic with 16,384 $= 2^{14}$ nodes can be implemented either as a 128×128 two-dimensional routing mesh or a $32\times32\times16$ three-dimensional routing mesh. The minimum bisection bandwidth of these two networks is the same: 128×80MB/s $=$ $16\times32\times20$MB/s $=$ 10.24GB/s (in each direction). The significance of this figure of merit is that if message destinations are selected at random (a worst case), then half of the messages must traverse the bisection. Unless a substantial amount of internal buffering is available, the network becomes saturated at approximately half the bisection capacity.

The usual argument that the bisection limits the total volume of messages that can be produced and consumed by the nodes applies only to the case of randomly selected destinations. For a 16K-node network, either 2D or 3D, this limit is 1.25MB/s per node, or, for a typical message length of 20 Bytes, an average of one message each 16μs. In fact, simulations of the Mosaic runtime system's process-placement strategies show that the localization achieved in process placement reduces the number of messages that cross the bisection to substantially less than this worst case. It may well be possible for nodes to produce and consume 20B messages at rates in excess of one message each 4μs.

Analyses that assume the worst case of randomly selected message destinations favor a higher dimension network than is necessary for more localized message traffic. Our original plan for the Mosaic was to use a $32\times32\times16$ three-dimensional routing mesh; however, it now appears that we will be able to save time and reduce risk by using a 2D network.

The latency using cut-through (wormhole) routing for a packet that is not blocked in the network is $T_{\mathrm{CT}} = T_p D + L/B$, where T_p is the path-formation time through one router, D is the distance, L is the message length (*eg,* in Bytes), and B is the channel bandwidth (*eg,* in MB/s). For a 20Byte packet, the L/B term is 1μs for the 3D synchronous router and 0.25μs for the 2D asynchronous router. T_p is two clock periods, or 0.05μs for the 3D synchronous router; the longest path through this network is $D_{\max} = 31+31+15$, so the maximum path-formation time is 3.85μs. T_p is expected to be 0.022μs for the 2D asynchronous router and the maximum path is $D_{\max} = 127 + 127$, so the maximum path-formation time is 5.6μs. In fact, for localized messages or longer messages (such as are encountered in program loading), the 2D network outperforms the 3D network.

Given the similar performance of these two networks, there are several other arguments in favor of using the 2D network:

1. The asynchronous 2D network eliminates the problems of coherent clock distribution required by the synchronous 3D network.
2. The protocol for the asynchronous 2D network is identical to that used in the Symult S2010 medium-grain multicomputer and the Intel Touchstone Delta prototype; thus, we would be able to employ the same host interfaces and other special devices (*eg,* displays) on either type of system.
3. The 2D packaging is considerably simpler, cheaper, and lower risk than the 3D packaging, and reduces the number of interboard connections by nearly a factor of four.

There is also an interesting issue of network scaling as it relates to our research agenda. The bisection argument presented above shows that the scaling of a mesh or torus network of given dimension is forced to the next higher dimension only when the radix (number of nodes on one dimension) becomes too large. The actual numbers show that 128 is close to the practical limit for the radix. Thus, if we can demonstrate that a 128×128 network and the localization accomplished by our runtime system still allow efficient execution with fully automatic process placement, we have also demonstrated that efficient execution would scale readily (with the problem size also scaling) to an $N = 128\times128\times128 = 2^{21}$-node system!

Another part of our long-term research agenda is to consider whether the third dimension should be reserved not for another dimension of mesh, but for long-distance connections; for example, a free-space optical shuffle. This consideration adds additional hesitancy to using the third dimension prematurely.

2.1.4 The Memoryless Mosaic chip

The *Memoryless Mosaic* chip has been a key part of our system-development strategy for the Mosaic. This chip (see the plot on the following page) is a complete Mosaic element except for the ROM and dRAM. It includes the Mosaic processor, packet interface, router, clock driver, and bus arbitration logic. The address and data buses are brought off of the chip; thus, the Memoryless Mosaic chip has allowed us to test the logic sections of the Mosaic under conditions in which the memory address and data are observable, and the memory data are controllable. It would otherwise be extremely difficult to diagnose internal problems in the Mosaic node, because the router, packet interface, and processor must function correctly in order to test them!

Extensive testing uncovered a design error in November 1989 in the first silicon of the Memoryless Mosaic, which was fabricated by MOSIS in 1.6μm SCMOS. The

Memoryless Mosaic chip

bug was in the packet-interface section, and was eventually traced to a missing $4\lambda \times 4\lambda$ patch of first-metal on one of the clock lines. This bug was not discovered during switch-level (Cosmos) simulation because the clock was supplied through an alternate path via a poly wire. This kind of error would ordinarily be expected merely to limit the speed of correct operation. However, in the Mosaic chip, it caused the control signals derived from the supposedly non-overlapping clock phases to overlap. The clock phases are generated on-chip, without the possibility of adjusting the non-overlapping time. As a result, several shift registers in the packet interface failed to operate correctly at any frequency. The detailed study of the FIFO section of the packet interface revealed ways of making it more robust, so this section was redesigned.

The corrected chip was submitted to MOSIS for 1.6μm SCMOS fabrication on 8 January 1990, and the revised parts were received on 14 March 1990. Preliminary tests indicate that the problem with the packet interface has been corrected, and the chips are fully functional.

To test the logic sections of the Mosaic in the target 1.2μm SCMOS technology, a Memoryless Mosaic with a new pad frame was submitted to the MOSIS 1.2μm SCMOS run that closes on 20 March 1990.

2.1.5 Program-development systems

The other important application of the Memoryless Mosaic chip is to accelerate porting the programming systems, particularly the operating and runtime systems, from simulators to hardware. This bootstraping step is on the critical path of developing a useful system, and is also typically more difficult for multicomputers and other distributed-memory systems than it is for shared-memory systems. The observability and diagnosis of operating-system faults is problematic until the operating system is itself reliable.

We are able to get a head start on porting programming systems and application programs to the hardware, and also to simplify the operating-system-porting task, by building program-development systems that are based on the Memoryless Mosaic chips. These 6U VME boards (see the illustration on the following page) include 4 Memoryless Mosaics, which are connected by their channels in a 2×2 mesh. The external memory of each of the Memoryless Mosaics is 128KB of SRAM, which is two-ported to be read and written either by the Mosaic or through the VME interface. The clock rate is 20MHz. The SRAM is accessed by the Mosaic most of the time, and by the VME interface by cycle stealing. When the VME interface requests a memory access, the clock generator PAL stops the Mosaic clock signal for one clock period. While the Mosaic clock is stopped, the VME memory access is granted. The Mosaic clock and reset can also be controlled by memory-mapped storage locations. The logic design of these VME boards was just completed, and they are being sent to a commercial PCB house for layout and fabrication.

Mosaic program-development board

The completed boards will be plugged into the VME interfaces of our Sun workstations or Symult S2010 systems. The host system will not only be able to load the memory of the nodes directly, but can also monitor program execution by examining the memory contents.

We expect in approximately three months to build another version of this program-development system for Memoryless Mosaics that use the asynchronous router. It will be possible to connect these boards together to form larger meshes, and to use these boards as host interfaces for larger Mosaic systems.

2.1.6 Packaging

Preliminary packaging designs for both 2D and 3D Mosaic systems have been completed. Both approaches use compression connectors to connect small circuit-board modules that are the testable and interchangeable units of manufacture, repair, and replacement. The Mosaic elements will be packaged and connected to the small circuit boards using TAB packaging.

The 4.2in$\times$2.6in module for the 3D Mosaic contains 8 nodes in a $2\times2\times2$ configuration with 320 external connections on two opposite edges. These modules are stacked between motherboards to create the 3D-packaging configuration. The 3D system is cooled by forced air in a direction parallel to the second dimension of routing.

The 4.2in$\times$4.2in module for the 2D Mosaic contains 16 nodes in a 4×4 configuration with 400 external connections on all four edges. These modules are mounted to a power-distribution frame, and adjacent edges are joined by a single bridging connector.

2.1.7 Programming systems

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. Although Cantor was developed specifically for programming the Mosaic, Cantor programs can also be run today on medium-grain multicomputers, multiprocessors, sequential computers, and the Mosaic simulators.

The Mosaic can also be programmed at a lower level by using scaled-down versions of the C-based programming systems (Cosmic C, Reactive C) that we have developed for and used with medium-grain multicomputers.

These programming systems are quite stable and powerful. The continued improvement of these systems depends principally on progress in our related research efforts (see sections 3.1–3.4).

2.2 Second-Generation Medium-Grain Multicomputers*

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

Our principal current research efforts with medium-grain multicomputers are aimed at new versions of our reactive-process programming systems and at advances in the performance of our mesh-routing chips. Our Caltech project continues to work closely with the DARPA-supported Touchstone project at Intel Scientific Computers. Our contributions include the architectural design, message-routing methods and chips, and system software. (See section 3.3 for a summary our current efforts with the Cosmic Environment and Reactive Kernel systems, and section 4.5 for a summary of our efforts with mesh-routing chips.)

The project operates several multicomputers: 8-node and 64-node Cosmic Cubes, a 128-node Intel iPSC/1, a 16-node Intel iPSC/2, and 32-node and 192-node Symult S2010 systems. The 192-node S2010 system is now the preferred machine for users. It is accessed through the Caltech Concurrent Supercomputer Facilities, and utilization has been at a level of approximately 90% of the available node-hours. All of these systems run very dependably.

Copies of the Cosmic Environment system have been distributed on request to approximately ten additional sites during this period, bringing the total copies distributed directly from the project to over 200.

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

3. Concurrent Computation

3.1 Runtime Systems for Fine-Grain Multicomputers

Nanette J. Boden, Chuck Seitz

We have been investigating several research problems that have emerged from our efforts to develop runtime systems for fine-grain multicomputers such as the Mosaic. These efforts are aimed at removing a number of restrictions on programming fine-grain multicomputers.

One easily understood example is the management of the node receive queue. A computation executing on the Mosaic will always consume a certain amount of space in each node for the runtime system itself, process code, process tables, and the persistent variables of the processes. The remaining space, which might be only one thousand bytes or so, can be used by the send and receive queues. Suppose that the computation involved a temporary "hot spot" that causes the receive queue in a node to overflow. When processes are able to exercise discretion in receiving messages selectively by their type or contents, they may not be able to consume the contents of the receive queue. In the present runtime systems, this is a deadlock, and the computation terminates.

It is, however, a serious flaw if a system with 1GB of memory, perhaps hundreds of MBs unused, might not be able to proceed because of a *local* fluctuation of a few hundred bytes. This problem also exists in medium-grain multicomputers, but is generally masked by the large size of the node memory. The solution is to export a part of the receive queue temporarily to another node, and, if necessary, to secondary storage. Indeed, several possible advances in system robustness and performance depend on introducing distributed solutions to resource-allocation problems.

Adding this kind of robustness to multicomputer programming systems is an example of the 80/20 rule: 80% of the sophistication in a runtime system is required to deal with the 20% residue of "difficult" cases and programs. Indeed, the compilation and runtime algorithms and heuristics for managing space without undue restrictions on the programmer, automatic process placement, managing the process-name space, determining code placement, and performing automatic code partitioning are remarkably subtle. They are also quite challenging when they must be implemented under serious constraints on both execution time and storage space.

Fast, efficient process placement is the key to several of these problems. Through analytical methods and simulation, we are exploring the spectrum from randomized to systematic node selection, that is, from methods depending entirely on randomization to methods that bias a random choice toward a local region or direction of growth, to methods that perturb a deterministic choice with "flip bits," to purely deterministic methods. A computation can be modeled for these purposes as an evolving population of processes. Each process on each timestep has a certain probability of creating another process or of self-destructing. Simulation approaches permit a realistic complexity in the algorithms and heuristics being evaluated, and the incorporation of realistic machine models. However, these investigations are still somewhat removed from reality. Different resource allocation strategies may be more nearly optimal depending on the actual characteristics of application programs. In the analytical approach, the probabilities of process creation and of process self-destruction must be estimated; in simulation, randomized instances of "typical" programs must be used as input. The Mosaic system will allow us to refine the more promising approaches on full-scale application programs.

3.2 Composition Properties of Reactive-Process Programs

Nanette J. Boden, Chuck Seitz

The properties of adaptive-routing message systems, which may appear in future multicomputers, have numerous implications at levels ranging from the programming model to the the runtime support. The most attractive distributed approach to retaining message-order preservation is based on a reply-message protocol. It happens that this approach introduces a slightly stronger synchronization than the semantics supported in our current message-passing programming systems, in which message order is preserved only between pairs of communicating processes. The reply-message protocol allows the sending process to determine when a message is actually in the receive queue of the destination process, so that subsequent messages to "third parties" cannot lead to messages that precede the first message in the receive queue.

This stronger form of synchronization also has composition properties that are more uniform than those exhibited by our present message semantics. Curiously, it is also possible to obtain uniform composition properties by weakening our present message semantics into the unordered-message form of Actor semantics, but we can show that at least a weak form of message-order preservation is required to express certain computations efficiently. Uniform composition properties are not only desirable when attempting to reason about a program, they are also critical for being able to re-express a large process as a collection of small processes, either by hand or automatically. We are continuing to study the possibility of supporting this stronger (but compatible) form of message-order preservation in future systems.

3.3 The Cosmic Environment and Reactive Kernel

Wen-king Su, Jakov Seizovic, Chuck Seitz, Joe Beckenbach, Christopher Lee

Our plans for the development of new versions of the Cosmic Environment host runtime system and the Reactive Kernel node operating system were outlined in our previous semiannual technical report, and the work is in progress.

Version 7.2 of the Cosmic Environment has matured after enduring more than two years of academic and commercial applications. Based on our experiences with the Cosmic Environment, we are now in the position to suggest and implement major changes in the internal structure of the Cosmic Environment. One of the problems in version 7.2 is the centralized multicomputer allocation and bookkeeping mechanism that places the Cosmic Environment at the mercy of network conditions. We have designed a robust distributed mechanism in which allocation is performed in the host of the multicomputer itself. Thus, the multicomputer would be inaccessible only when its host is inaccessible. We have also demonstrated a technique that increases the Cosmic Environment communication bandwidth from 40Kbytes/second to 300Kbytes/second with a small increase in message latency. We eliminate the need to perform extra handshakes across slow ethernet links by shifting the burden of buffering messages from the multicomputer's host machine to the user's host machine. We have also found a way to increase the message delivery rate for selected user processes, such as a frame buffer controller, by allowing the process to be merged with the message switcher process, thus saving one communication cycle and context-switch time for each message.

3.4 The Page Kernel

Craig S. Steele, Chuck Seitz

The previously-described "Page Kernel" (PK) concurrent programming environment is an evolutionary variant of the reactive kernel (RK). PK utilizes the virtual-memory capabilities of second-generation medium-grain multicomputers to render message origination and receipt implicit, and to move the low-level management of data sharing from the programmer to the kernel. Continuing development of the PK has resulted in simplification of the programming model and extension of its capabilities.

The executable unit is the *action*, a light-weight reactive process scheduled in response to modification of associated data structures (*blocks*). The programmer is responsible for writing code to specify which data blocks are accessible to each of the actions. Defining the multiple address spaces of the actions and coding the operations of the actions is the programmer's task; action scheduling and data communication are handled by the kernel.

Another common function appropriated to the kernel is the management of mutually-exclusive writing to data blocks shared by multiple actions. Rather than locking data with potential write conflicts, actions are allowed to proceed to completion before actual conflicts are evaluated. If an action is excluded from writing its results to a shared data block due to another action's access, it fails and none of its results are written to any data block. The action is undone with no visible effect, and it is rescheduled for later execution. This mechanism involves considerable data copying and duplication, but the additional cost is quite modest with second-generation multicomputer communications hardware; for example, it incurs about 25% in increased execution time on the Symult S2010. This implementation allows greater concurrency for problems with more potential than actual conflicts.

The PK is expected to be an attractive alternative programming environment for problems such as iterative optimization, in which the mechanics of distributing and updating shared data structures may obscure the relative simplicity of a concurrent algorithm.

3.5 A C-Based Concurrent Programming Language For Multicomputers

Marcel van der Goot, Alain Martin

As described in the previous semi-annual report, we are defining and implementing a concurrent programming language for message-passing multicomputers. We have chosen C as the basis for the sequential parts of the language; the extensions that support concurrent programming include processes and CSP-like communication primitives. A first implementation, consisting of a compiler and a small runtime system, was finished in February 1990. The compiler takes our language as input and has standard (ANSI) C as target; the runtime system contains functions to support the concurrent execution of processes. The output of our compiler is compiled for a SUN workstation where it is executed as a single UNIX process.

So far, the compiler has been used by the students in a concurrent programming class, and to write a (functional) simulation of the asynchronous microprocessor. Since the specification of the microprocessor is in a language similar to ours, the simulation program was relatively easy to write. Currently, we are working on documentation and on porting the implementation to an actual multicomputer (the Symult S2010, or any other multicomputer that runs CE/RK), together with some reorganization of the compiler. We expect that neither the compiler nor the runtime system will require much rewriting for this parallel implementation.

4. VLSI Design

4.1 Automatic Synthesis of Asynchronous Circuits

Dražen Borković, Steve Burns, Alain J. Martin

The second generation of synthesis tools that we envision will integrate simulation, performance evaluation, and optimization (transistor sizing). The designer will be able (or perhaps will be required) to make choices at different stages of the synthesis based on the results of the previous stage. As a first step toward such a system, we are designing a program for the synthesis of straightline program into CMOS chips. The final program will include automatic cell synthesis, transistor sizing, placement and routing.

4.2 Cache Memory for an Asynchronous Microprocessor

Alain J. Martin, José A. Tierno

The design of a direct-mapped instruction cache for an asynchronous microprocessor is almost completed. The circuit has been derived from a high-level specification, and both control circuitry and RAM array are completely delay-insensitive with the exception of isochronic forks. Special attention was paid to the design of the RAM cell, to optimizing the signaling protocol, and to eliminating unnecessary transitions and completion trees. The full (conservative) implementation requires 13 transistors per memory cell, of which 3 can be eliminated at the expense of a bigger delay. The RAM array has a special read-write cycle. The rest of the control was designed around this cell, since the bottleneck in throughput will be in the access to the RAM array.

4.3 Testing Self-Timed Circuits

Pieter Hazewindus, Alain J. Martin

We are studying the problem of increasing the fault coverage of our designs by adding testing circuitry to the circuits. The fault model we use is the single stuck-at fault model. For any non-redundant circuit, if we can set and observe the value of each state-holding element, then all faults are testable. Since it is infeasible to connect every state-holding element to a pad, we use as testing circuitry a simple queue that connects all state-holding elements. For such a scheme, the only untestable faults would be located in the queue.

We have designed a testing queue that has twelve transistors per stage. For normal circuit operation, the penalty for having the testing circuitry is just one pass gate, so that the decrease in performance is minor. For the control of the microprocessor, the number of transistors in the clocked testing queue is about half the total number of transistors. We are trying to reduce the size of the testing queue by reducing the number of state-holding elements observed. It seems that possible global optimizations, at the program level or otherwise, are rare, but some ad hoc or local optimizations are possible.

4.4 Sizing the Transistors of Asynchronous Circuits

Steve Burns, Alain Martin

We have developed a method of optimally sizing the transistors contained in the asynchronous circuits that we construct by systematic transformation from concurrent programs. These transistors are sized optimally if the sizes minimize the time needed to operate the circuit, minimize the energy required to operate the circuit, or minimize some other metric of performance.

The concerns of performance optimization in asynchronous circuits are quite different than those of synchronous (clocked) circuits. In the synchronous cases, the main task is to determine and then speed up the slowest or critical path through the combinational logic that connect the clocked latches. This is in order to maintain correctness, since for correct operation, the combinational logic must complete before the clock changes.

In the asynchronous circuits derived using our synthesis method, the circuits work correctly regardless of delays in the primitive gates. For most applications (i.e., those without hard real-time deadlines), it is not necessary to optimize the worst case (or even to know what it is). Rather, it is the average case that determines a circuit's performance. While an operation that requires twice the time but occurs only once every one hundred operations is catastrophic to a synchronous design, it only decreases the performance of our asynchronous circuits by one percent.

Much of the computation involved in the performance analysis of synchronous circuits, in particular that of determining the critical paths induced by unusual data patterns, can be avoided by using our asynchronous methodology. An average or typical operation sequence is specified and a performance metric is determined based on that sequence. Since our asynchronous circuits work correctly regardless of gate delays, it turns out that the performance metric is a convex function of the transistor sizes and thus each local minimum to the function is also a global minimum. The techniques of convex non-linear programming can be used to find these optimal sizes. A C program has been written to perform these calculations. Optimal transistor sizes for a typical 40 transistor circuit can be obtained in under 10 seconds on a SUN 3/60.

4.5 Fast Self-Timed Mesh-Routing Chips

Chuck Seitz

A new version in the FMRC series of mesh-routing chips has been laid out, verified by switch-level simulation, and sent to fabrication for the 1.2μm MOSIS SCMOS run that is scheduled to close on 20 March 1990. Previous FMRC chips have been fabricated in 1.6μm SCMOS, and operate at 65MB/s, but exhibit some reliability problems when the aggregate throughput of the chip's 5 output channels exceeds about 250MB/s. This reliability problem was traced by analysis and simulation to collapse of the internal power supply under these demanding conditions; thus, it is properly a failure of the packaging rather than of the chip design.

This 132-pin chip devotes the 20 lowest-inductance PGA-package pins to Vdd and GND. It was not deemed to be practical for the immediate application (the Intel Touchstone Delta prototype) to increase the pinout to allow additional Vdd and GND pins; however, it was considered to be desirable to increase the speed to in excess of 80MB/s. Intel is tooling a special package whose internal power and ground planes reduce the inductance of the power distribution from the package by a factor of approximately two. However, in designing new pad circuits and pad frame for the FMRC, I decided to take all available measures that might improve the reliability of these chips.

With the support and encouragement of Wes Hansford at MOSIS, we were able to reduce the pad pitch from 6 mils to 5 mils, with a 90μm square pad. The resistance of the pad-power ring was reduced in comparison with our standard 1.6μm pads by a factor of nearly four by a combination of increased width and use of both metal layers where possible. The peak pad-drive current was reduced to about 0.75 of its value for the 1.6μm pad drivers, and the p/n ratio was reduced from 5/3 (which produces symmetrical transitions in the 1.6μm process) to 4/3 to compensate for the transistors being farther into velocity saturation. Additional speed in the core of the router will more than make up for the slightly slower pads. These measures reduce the total current and ohmic drops; they also decrease di/dt effects of the package-pin inductance. As additional measures to reduce the di/dt effects, nearly all of the "white space" in this pad-limited design was used to add power-decoupling capacitance, which is believed to be more than 500pF. The drive of the output pads was also tuned to minimize di/dt. (A plot of the chip is shown on the following page.)

The design and layout of a successor to the FMRC is underway.

4.6 Adaptive Routing in Multicomputer Networks

Mike Pertel, Chuck Seitz

Previous theoretical studies of adaptive multipath routing are being continued, and an adaptive router for the Mosaic is being designed. Under simulation, adaptive routers have exhibited superior throughput, traffic diffusion, and fault tolerance, as compared with oblivious routers. Further simulation is being used to refine and simplify the routing discipline before committing to silicon.

4.7 High-Density Mosaic dRAM

Don Speck

FMRC2.2 mesh-routing chip

Multicomputers have been tending toward more memory per node as they get faster, and Mosaic is no exception. Having more never hurts, and it extends the application range and ease of programming. Therefore, when the Mosaic C design began, design of a dense dynamic memory began with it. The simulation and layout of a 32K$\times$16 dynamic RAM is now complete, and ready for first fabrication in the $\lambda = 0.6\mu$m MOSIS SCMOS process. This 64KB memory is half as much as in a Cosmic Cube node, and is the largest power-of-2 size smaller than Mosaic C's addressing limit. It is also the largest area ($13470\lambda \times 11974\lambda$) that doesn't need repeaters in all of the wires, and is about 75% of the total chip area (which is how much was budgeted for RAM).

The design of this dynamic RAM attempted to simultaneously optimize area, energy, speed, and noise immunity. Small area is the primary reason for choosing a one-transistor-per-bit style instead of something easier to analyze (otherwise why bother?), and it also helps shorten the long wires that contribute to delay and power consumption. Power dissipation is at a premium in large ensembles of closely-packed nodes, and the only way to significantly reduce total chip power is to reduce the power supply voltage to 4V or even 3.3V; for wafer-scale packaging, 2.5V would be required. In addition, a safety factor of plus or minus 20% is needed to allow for process variations. Over such a wide operating range, it is not possible to meet a fixed speed and noise immunity specification regardless of voltage, nor is it necessary. The RAM only has to keep up with the processor, whose speed varies with voltage, and the noise immunity has to exceed noise generation, which also varies with voltage (quadratically in the case of resistive drops, less than linearly for the backgate component of threshold variation).

To accommodate the processor on the same chip and have access to the smallest feature size of the day, the RAM uses a standard MOSIS logic process and is designed to satisfy all of the Magic DRC rules for the most restrictive process, in either nwell or pwell. (The latter disallows boosted signals). The best bit storage capacitor in that process is an enhancement-mode MOS capacitor, which has low charge-storage density and cannot store the full power supply voltage range. These are the same limitations that the early commercial dRAM designers faced, so the support circuits that worked well then also turn out to be good choices for this RAM.

Making the cell capacitor large to compensate for low charge-storage density is subject to diminishing returns. The bitline length and capacitance grow with the cell capacitor. Larger depletion regions collect more minority carriers from alpha-particle strikes. Larger MOS capacitors are slower and cannot be charged as fully in the time available; even with a modest capacitor size, writing has to start very early to approach full charge. Beyond some point, the area is better used elsewhere, such as for more sense amps, and this point is about $64\lambda^2$. This is just big enough for a half-sized dummy cell to be feasible. A full-sized dummy cell would need a half-charge reference voltage, which is not $V_{dd}/2$ due to the MOS capacitor threshold. At the lowest operating voltage, the capacitor cannot even store $V_{dd}/2$.

The small bitcell has room for only one bitline through it, and, without a second poly layer, this mandates an open bitline arrangement. Open bitlines require more careful matching of noises on opposite sides of the sense amp than do folded bitlines. There is no place to put transistors to short together bitline pairs; instead, oversized prechargers short all bitlines to an equilibration line, which then connects to Vdd only at its center tap, to equalize power glitches. The substrate has similar equilibration wires center-tapped to ground spaced 16 bitlines apart, taking up about 5% of the RAM area.

These noises can't be perfectly matched, so it is advisable to make the readout voltage large in comparison, in this case by keeping the bitlines short — only 32 bitcells — resulting in a 6:1 bitline-to-cell capacitance ratio. The sense amplifiers have to be small and simple to avoid dominating the total area, but a simple cross-coupled pair suffices when the signal voltage is large and bitline capacitance is low. Low bitline capacitance also makes full-V_{dd} precharge affordable, which is needed anyway because at the lower supply voltages (*eg,* 2V), $V_{dd}/2$ precharge wouldn't be enough to turn on the sense-amp transistors. The column-select transistors double as cascodes that isolate the bitlines from the I/O line capacitance until the bitlines fall a threshold below V_{dd}. Area-consuming level-restore circuits are not needed on the sense amps, because the storage capacitor cannot store full voltage levels, but one is used on the I/O lines in case the bitlines fall far enough for the cascodes to slowly leak.

There are 8192 sense amps but only 16 bits need be read or written at once. There is neither need nor room for a read/write amplifier per sense amp. Fortunately, the bitline pitch is larger than minimum metal spacing, leaving enough room to intersperse column select lines from a shared column decoder, controlling the multiplexing of 64 sense amplifiers onto 2 read/write amplifiers via I/O lines perpendicular to the bitlines. Space has to be made periodically for read/write amplifiers to keep the I/O line capacitance low enough to be driven quickly by the sense amplifiers, providing a good place to insert row decoders that keep the wordlines short enough to run in poly without metal strapping. Strapping the wordlines would have increased bitline capacitance by 10%; the increase in bitcell area needed to counteract this would have been more than the row decoder area.

The short bitlines and wordlines divide the RAM into 8 by 8 banks. To keep each data bus wire under 12000λ, only 2 bits connect to each bank, so 8 banks must power up on each cycle. About half of the power consumed goes into address distribution, decoding, and clocks. If prechargers in unselected banks were turned on and off every cycle, that would add 25% to the power consumption (all from the clocks); instead, the first three address bits control them. Precharge turn-on needs to wait anyway until the wordlines finish falling; hence, it is controlled by a delay line. This obviates any need for a second clock phase, saving clock wiring and its attendant power dissipation.

The sense amplifiers are on a 10.5λ pitch; this demands that they be connected common-source to a current generator. The amount of current a sense amp receives depends both on its own bitline voltages and on the bitline voltages of other sense amps. Initial current is set low, so that sense amps receiving the most current get no more than is safe, although this means that some sense amps receive none at first. As the sense amps with an early start develop signal, current is ramped up until all sense amps are conducting. Further current increases are delayed until the late starters catch up, then a larger current ramps up. The sense timing generator ramps up voltages on transistor gates via current mirrors, and fits underneath the row decoder address wires along with a delay line to simulate the wordline delay.

```
AREA BREAKDOWN:
    bitcells 61%
    sense amps, prechargers, dummy cells 15%
    power/ground wires 11%
    row decoders 8%
```