Submicron Systems Architecture

Semiannual Technical Report

Computer Science
California Institute of Technology

5178:TR:85

March 1985

Reporting Period:	16 October 1984 to 15 March 1985
Principal Investigator:	Charles L Seitz
Faculty Investigators:	James T Kajiya
	Alain J Martin
	Robert J McEliece
	Martin Rem
	Charles L Seitz
	Henk Van Tilborg

Sponsored by the
Defense Advanced Research Projects Agency
ARPA Order Number 3771

Monitored by the
Office of Naval Research
Contract Number N00014-79-0597

SUBMICRON SYSTEMS ARCHITECTURE

Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document reports the research activities and results for the seven month period 16 October 1984 to 15 March 1985 (5 months) under the Defense Advanced Research Project Agency (ARPA) Submicron Systems Architecture Project.

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, and includes related efforts in concurrent computation and VLSI design. Additional background information can be found in previous semiannual technical reports [5052:TR:82], [5078:TR:83], [5103:TR:83], [5122:TR:84], 5160:TR:84.

1.3 Highlights

Some highlights of the previous 7 months are:

- Cosmic cube technology transfer (2.1.1).
- First working Mosaic RAMs (2.2).
- Concurrent Prolog process model (3.1).
- Denotational Semantics for Prolog (3.1).
- New method for compiling programs into self-timed systems (3.6).
- New results in burst error codes (4.1).
- First benchmarks on the MOSSIM Simulation Engine (4.3).

2. Architectural Experiments

2.1 Cosmic Cube

W C Athas, Reese Faucette, Wen-King Su, Chuck Seitz

2.1.1 Technology Transfer

We reported in our last semi-annual technical report that a non-exclusive license had been negotiated for the cosmic cube, covering design and patent rights, and a resale license of the operating system.

We are now at liberty to report that this licensee is Intel Corporation. Intel announced the "iPSC" (Intel Personal Supercomputer) line of computers, versions connected as 5-cubes, 6-cubes, and 7-cubes (32, 64, and 128 nodes, respectively), on 11 February 1985, with first deliveries scheduled for June 1985. These systems are about 4 times as powerful per node than the cosmic cube. The instruction processor is an Intel 80286/80287 rather than the 8086/8087, the storage is 512K bytes rather than 128K bytes, and the communication channel rate is 10 Mbits/s instead of 2 Mbits/sec.

The technology transfer effort has been relatively straightforward. The Intel machines are expected to be very nearly (C) program compatible with the cosmic cube in spite of hardware differences, due to the resident operating system hiding hardware details successfully.

The terms of the license agreement include a contribution of an iPSC d7 (7-cube) machine that will be operated by the Caltech Submicron Systems Architecture Project. This machine will be available on the ARPAnet for programming and application experiments.

2.1.2 Retrospective and Future Plans

The cosmic cube project has reached a stage in which system-building has yielded to the fascination of programming and using this computing system. It is more an more being treated as a part of our regular computing environment. Thus it is time to offer some reflections on what we have learned from this architectural experiment, and to report where we plan to go from here.

We are quite pleased with the computational model and the corresponding primitives provided by the cosmic cube's operating system, the "cosmic kernel". We have come to consider them to be fundamental to ensemble machines of this grain size.

The queued communications with no acknowledgement of message receipt is a very weakly synchronized form of communication. When this property is strengthened with message order being preserved between any pair of processes, the message primitives are (1) very well suited to a broad class of typical programs, and (2) capable of expressing, if needed, the strongest forms of synchronization.

Another crucial property of the message primitives is that message sending and receiving is handled as a concurrent activity of a process. Hence, a process can manage the sending and receiving of many concurrent messages.

Other features of the message system are motivated by portability: (1) there is just one format for messages of any length, whatever the underlying protocols; (2) messages are sent and received whole; and (3) the semantics of the message operations are independent of process placement. Thus the expression of a concurrent computation for the cosmic cube is quite general and is reasonably portable between different machines, concurrent or sequential, that support this same multiple process message passing environment.

Future development is expected to follow along two general lines.

We plan to migrate some of the lower level functions of cosmic kernel into a custom communication chip. From what we have learned in the design, implementation, and use of the cosmic cube message system, we now see way to manage the flow control and routing of messages that reduces the average communication latency and also lends itself very well to a VLSI. Work already has started on a chip called the "wormwhole chip" to handle message routing inside a cosmic cube, thus relieving the processor of this task. This chip uses the same e-cube routing algorithm that is used in the cosmic cube. Unlike the current system, which reads a message completely in before forwarding it, the wormwhole chip moves a message a flow-control packet at a time through a node. Thus the path of a message is analogous to a worm weaving its way across the machine. The worm's head can be at the destination while the tail is still entering the network. There is no queueing inside the routing chip aside from the single packet that is buffered in each channel of the chip, so all queueing must be accomplished at the source and destination. This technique has been simulated for much higher message rates than are typical of cosmic cube programs, and for cubes as large as 12 dimensions, in a program that runs on the 6-cube.

Second, we plan to use the primitives of Cosmic Kernel as building blocks for designing new programming languages for the cosmic cube. Our current efforts have been to provide a C programming environment, since as a system's programming language, C meshes well with the cosmic kernel primitives. A comprehensive C user's guide to the cosmic cube has been written, as well as many C programs. However, looking at message passing actions not as system calls, but as constructs of a programming language, raises many questions and suggests a number of opportunties. The cosmic cube's message operations, although included simply as C functions, actually create a concurrent process with shared variables. The descendant process persists until the specified message operation is completed.

Program flow analysis, such as live variable analysis, can be applied even to our present C programs. The result of program flow analysis would permit the automatic insertion of the code necessary to test message locks as well as the automatic setting up of message descriptors, thus relieving the programmer from a good deal of the tedium associated with such functions. Experience with this approach to the general problem of how to express concurrent computations within this process model is leading to a concurrent object-oriented programming model that may well justify the design and implementation of a programming language particularly for such machines.

2.1.3 Hardware and Software Status

Our 6-cube (64 node) and 3-cube (8 node) Cosmic Cubes continue to run reliably, the last hard failure having occurred 10 months ago. With at this time a total of over 1,000,000 node-hours of operation and only two hard failures logged, both in dRAM chips, the calculated MTBF of the nodes of 100,000 hours can be regarded as conservative at the 99.9% confidence level, and an MTBF of 200,000 hours can be stated at a 95% confidence.

A second of the "rosebud" type of Cube Life Support board was built so that the interface to the 3-cube and 6-cube are identical. The current 6-cube host is a SUN file server, ARPAnet address `cit-sol`, and the current 3-cube host is a SUN workstation, ARPAnet address `cit-neptune`.

Both machines continue to be heavily used, even though all the Caltech physics department use has, as planned, been phased out.

The cosmic cube's resident operating system, the "cosmic kernel", is complete, but continues to be refined, particularly in the areas of performance tuning, debugging and error handling features, and the host software. Documentation of the system for users is also complete, and also continues to be refined.

With the cosmic kernel and host tools now quite stable and reliable, we are planning to experiment with time- and space-sharing of the 6-cube. The present "cube daemon" that runs in the SUNs as a user process will be enhanced to run as a system program that can serve multiple users.

2.1.4 Programming Techniques

Several programs intended to exempify and demonstrate programming techniques have been written, including:

- A broadcast spawning process, which, in addition to illustrating programming techniques, has become a "library" process to speed up program loading. (Faucette)
- A simulation of the time evolution of many-body systems that interact by symmetrical or unsymmetrical forces. (Seitz & Su)
- Programs to compute solutions to LaPlace's equation by relaxation, a classic program for parallel machines. (Su)

- A numerical sieve for the "Collatz problem", which differs from previous such programs both in using concurrency and by examining classes of integer expressions rather than individual integers. (Seitz)
- A simulation of the message traffic for "wormhole" routing in a binary n-cube, and using a different message flow control and routing technique (see 2.1.2) than is used in the cosmic cube. (Su)

Some other programs being written by the ARPA group include:

- A concurrent solution of the n queens problem. (Dally)
- A graphics package. (Su)
- Game playing programs. (Several)

Some of the programs being written for reasons other than illustrations of programming technique are described in later sections.

In addition, we continue to make the cosmic cube available to selected guest researchers for problems that appear to be particularly interesting or difficult, including in the last reporting period: guests from the Caltech engineering division doing jet plume computations, and guests from NSA.

2.2 Mosaic Systems

Steve Rabin, Don Speck, Chuck Seitz

Work is proceeding on schedule, in spite of a setback in our first Mosaic RAM chips, for assembling a 1024-node system based on a single chip Mosaic node. The principal effort in the past 5 months has been in discovering the sources and remedies for problems observed in the Mosaic RAM chips. These chips use a devious circuit design style known as "hot-clock nMOS", described in an attached paper.

The first prototype dynamic RAM module (later to be integrated with the Mosaic processor and communication channels) was received from MOSIS, tested, and found to have a geometrical design rule error that obscured the test results. The layout was fixed, and a second iteration was fabricated. The first chips from this second design iteration were able to read but not write. This problem was eventually traced to a clocked-isolation clock-AND circuit (see figure 9 in the attached paper) that were producing an output pulse when their *enable* input was 0.

Chips from the next MOSIS run, which has a fairly large t_{ox} of 65 nm and a very high enhancement threshold voltage of 1.2 volts, worked correctly, although not as fast as we had expected (7 MHz instead of 20 MHz clock rate). It is interesting to us that we obtained for this "out of spec" MOSIS run some design information that was vital to getting back on schedule with this project.

From this testing and from SPICE simulations, we have learned how to broaden the tolerance of these circuits to variations in process parameters over a range that should be completely safe for MOSIS processes.

The effort for the top assembly and packaging of the 1024-node Mosaic is underway.

3. Concurrent Computation

3.1 Concurrent Prolog Process Model

Pey-yun Peggy Li and Alain J Martin

Due to its expressive power and simple semantics, Prolog becomes a promising language for many applications. Moreover, logic programs are inherently well suited to parallel computations. A Concurrent Prolog Process Model is proposed to carry out OR parallelism, AND parallelism, and stream parallelism inherent in a Prolog program. The target machine of this Process Model is the Sneptree. Hence, distributed computation without shared memory and with message passing synchronization is assumed.

The Process Model contains two types of processes, AND processes and OR processes, which are alternatively connected into a proof tree. There exist communication channels between sibling AND processes to transmit the values of shared variables. The AND parallelism is realized by a data driven model and an input buffer merge mechanism. One major distinction of our model from previous models is that the processes in our model generate all the solutions at once, send them out to where they are requested, and terminate themselves immediately. A smart merge algorithm in the receiving process then combines and synchronizes the partial solutions from different places to form correct results. With this mechanism, we can avoid complicated backward execution in previous models, and thus more parallelism, less overhead, and simpler control can be achieved.

An extension of standard Prolog that integrates the concepts of Shapiro's Concurrent Prolog and Wise's Epilog is used as the base language in the Concurrent Process Model. An ordering algorithm is designed to construct the data flow diagram of the AND processes and detect the possible multiple paths between any two AND processes. Hence, the message channels between two AND processes or one AND processes and its father can be built up. Necessary synchronization signals are also generated by proper processes. A merge algorithm is also designed to either gather all the partial solutions from its descendants in an OR process or combine the input variable bindings from its sibling AND processes in an AND process. The merge algorithm does a Cartesian Product over all the input streams. With the appearance of synchronization signals, it does several Cartesian Products against the proper synchronization signals.

Mapping of our Process Model onto a Sneptree is going to be investigated next. Since a Sneptree can simulate an arbitrary size binary tree optimally, we expect that our model can be mapped onto the Sneptree with very little overhead.

3.2 Inverse Limit Construction of Infinite Lists

Young-il Choo, Alain J Martin

Infinite lists arise when we want to give meaning to non-terminating, yet answer producing, programs. For applicative languages this is possible using the lazy evaluation mechanism. Precise formulation of infinite lists is necessary for formal reasoning of such programs.

Infinite objects can be computed only as a limit of finite ones. The notion of a limit presupposes some kind of a topology, or at least an ordering. The ordering we use is that of definedness. By generating a sequence of finite lists that become more and more defined, we can specify an infinite list. This notion is embodied in a construction from topology called the inverse limit.

We have constructed a domain of infinite lists by taking the inverse limit of the chain of domains of finite lists ordered by projection. The domains contain finite lists up to a certain length, and the projection is that of the best approximation in the domain below. We have proved that the resulting domain is a complete partial order, and therefore, can be used as a semantic domain for the fixed point approach to program semantics.

These results will appear in a report "Notes on infinite lists I: Inverse limit construction".

Future work will explore the topological structure of the domain of infinite lists, and consider the connections with Scott's domain for λ-calculus which is also constructed using the inverse limit.

3.3 Denotational Semantics of Prolog

Young-il Choo, Jim Kajiya

The denotation of a Prolog program is a function (called substitution) that maps variables to terms of the Herbrand universe. The standard ordering defined componentwise on the variables makes incomparable those which differ at most by renaming of variables.

We have defined the ordering between substitutions that uses the category theory notion of one function factoring through another, i.e. $f \sqsubseteq g$ iff there exists h such that $g = h \circ f$. This enables a clean fixed point approach to Prolog type languages where the nondeterminism is taken care of by the function compositions.

Once the semantic domain is defined, we can verify Prolog programs by algebraic transformations.

3.4 Process Placement

Craig Steele, Chuck Seitz

This effort is directed at optimization of process placement on distributed homogeneous processor architectures, and related comparative studies of different direct networks.

A process model computation may be represented as a graph, where communicating processes are the vertices, and logical communication channels are the edges. Likewise the underlying physical structure may be represented as a graph with processors and physical communication paths represented as vertices and edges. To run a computation on a distributed processor, the processes must be loaded onto the physical machine, requiring a mapping of the logical graph onto the physical graph.

Minimizing the communication cost of the computation is a major problem unless the logical and physical graphs are similar. While many problems of interest can take advantage of the isomorphism of the n-cube to three-space, even in three-dimensional simulations the density of simulated objects may vary, lowering efficiency with simple partitioning. Graph structures common in applications of interest in computer science, such as trees, do not map in obviously good ways to n-cube architectures.

Using the technique of simulated annealing, good mappings may be found in moderate time for arbitrary logical computation graphs. Taken into account are arbitrary edge weights (reflecting varied utilization of logical communication channels) and arbitrary process memory sizes (allowing the physical processor constraints to affect the density of processes per processor).

This method has been applied to a comparative study of proposed interconnection structures for homogeneous machines for both single and multiple process to processor mappings. Efficient mappings are found for interconnects less costly than binary n-cubes for most problems at only modestly increased communication costs. For example, a modified shuffle-exchange interconnect for the 1024-element Mosaic design has been simulated for equally large problems to good effect.

Work is continuing to extend the architectural comparison to include congestion factors for networks in which communication channel saturation is significant.

Future work will include modification of the existing program to allow the optimized mappings it generates to direct loading of processes into the Cube. User-defined labels for vertices and edges will allow a more friendly style of specifying process connections.

Experiments to determine the convergence rates of the program using more localized cost functions and with informational 'noise' will determine the suitability of the method for concurrent implementation.

3.5 Concurrent Data Structures

Bill Dally, Chuck Seitz

The appearance of concurrent computers such as the Cosmic Cube and the Mosaic is creating a need for concurrent data structures and algorithms. While a great deal of work has been done on concurrent algorithms for numerical problems, very little is known about concurrency in non-numerical applications. Conventional data structures such as heaps and B-trees have many bottlenecks which limit their potential concurrency and make them unable to take advantage of the computing potential of these concurrent machines. New data structures are required to harness the power of concurrent computing.

Our research has been proceeding in two directions: to study the concurrency limitations of existing data structures and to develop new data structures which overcome these limitations. A heap was selected as representative of existing data structures and its concurrency properties were studied in detail [5156:DF:84]. As a result of this research some variants of conventional heaps were developed which offer improved concurrency and virtual memory performance. However, even these variant heaps are limited by their tree structure to concurrency which grows only as the log of the number of elements in the heap.

To overcome the concurrency limitations of conventional data structures we have developed two new data structures for ordered sets: the balanced cube and the B-cube [5174:TR:85]. By using a binary n-cube rather than a tree to organize data, these structures overcome the bottleneck of a single entry point or root node. As a result, the concurrency of these cube structures was thought to grow almost linearly with the number of elements in the set. A set of experiments has been run to test the balanced cube algorithms. The tests showed that the concurrency of the cubes grows $O(N/logN)$, not linearly as originally predicted. This sublinear performance is due to a mismatch between the topology of the cube and the order relation in the set. An obvious way to correct the mismatch is to add a ring connection to the binary n-cube computer; however the adjacency perserving mapping described below is a more attractive solution.

A novel method of mapping the elements of an ordered set to the processing nodes of a concurrent computer using a Gray code has been developed. [5176:DF:85]. The Gray code preserves adjacency. If the Gray code mapping is used, two elements which are neighbors in linear space are mapped to neighboring processors on the cube. Because of the reflection properties of the Gray code, a balanced cube using this mapping is also easier to search than a direct mapped cube.

Current work on concurrent data structures is concentrating on graph algorithms. Concurrent algorithms have been developed for the shortest path problem and for graph searching problems. Experiments to test these algorithms are in progress.

3.6 Self-timed Design

Alain J Martin

A paper describing a method for compiling a set of communicating processes into a self-timed circuit, including an illustration of its application to a distributed mutual exclusion algorithm, is attached.

3.7 Wormhole routing

Wen-King Su, Chuck Seitz

A new technique for flow control and routing in a binary n-cube was devised. In this technique no more than a flow control packet, typically 32 bits, is queued in a node of the n-cube. Messages flow directly through a node unless the required outgoing channel is already occupied, in which case the blocking of channels propagates back to the source of the message. In combination with the e-cube routing algorithm this technique can be shown to be free of deadlock.

However, because of the propagation of blocking, it is not intuitive that the performance of this routing scheme would be satisfactory, nor is the message flow analytically tractable. Some early simulations on an IBM PC suggested, however, that the performance of this routing scheme is excellent. A full scale simulation program was written for the cosmic cube, and run on the 6-cube for connections up to 12-cubes (4096 nodes). A more complete report will follow a series of simulation runs to gather complete statistics.

3.8 Multiprocessor Architectures for High Performance Computer Graphics

Daniel S Whelan, Jim Kajiya

My work during the last quarter has concentrated on the design and implementation of multiprocessor algorithms for producing images of synthetic scenes with shadowing effects. With the exception of some ray tracing architectures, no multiprocessor architectures proposed for computer graphics support shadowing effects. While ray tracing architectures produce extremely realistic images, they do so rather slowly making them inadequate for real-time simulation applications.

Shadowing is inherently a non-local effect, thus my work has concentrated on partitioning the shadowing computations such that interprocessor communications is both localized and minimized. My approach separates shadowing computations into two distinct processes which I call local shadowing and foreign shadowing.

Local shadowing determines whether any object within a processor's view space casts a shadow upon any other visible local object. Foreign shadowing determines whether objects outside the processor's view space shadow objects within the processor's view space.

My implementation of a foreign shadowing algorithm partitions well onto a multiprocessor with only very local interprocessor communications. The performance of this shadowing algorithm should far exceed the performance of ray tracing algorithms. The overall architecture will provide performance well in excess of that of today's flight simulation engines and will provide features such as shadows which today's flight simulators do not implement properly.

4. VLSI Design

4.1 Soft Error Correction for Increased Densities in VLSI Memories

Robert J McEliece, Khaled Abdel-Ghaffar, Henk Van Tilborg

In previous work, we have studued what might be called the "ultimate" physical limits of densities in VLSI RAMS, studying in detail the restrictions imposed by quantum and thermodynamic effects, for example. Recently, however, we have turned the more immediately worrisome effects of alpha-particles and other types of "soft" errors.

At sub-micron feature sizes, a single soft error event will in general cause not just a single bit error, but a two-dimensional "patch" or "burst" of errors, perhaps on the order of 2 - 4 bits square. Thus we have been focusing or most recent research on the problem of two-dimensional burst-error correcting codes, a subject which has until our own work not received much attention in the literature.

We have devised three new classes of two-dimensional burst correction codes, each of which is more efficient than any previously studied class. The first class is based on the new idea of "burst identification." A code in this first class is constructed from two simple sub-codes, one which is able to identify the burst pattern, and one which is able to locate it once it is identified.

The second class is based on extending the well-known Fire codes, which are successfully used to correct ordinary one-dimensional bursts. While the notion of "two-dimensional Fire codes" has appeared in the literature, our generalization appears to be superior to the older one, both from the standpoint of minimizing overhead and ease of implementation.

The third class is an ad hoc class of good two-dimensional cyclic codes found by computer-aided search. For the small burst sizes likely to be important in VLSI applications, this class actually appears to be the most promising.

Finally, we mention that as an unexpected by-product of our research, we have recently been able to prove a long-standing conjecture about one- dimensional cyclic burst error-correcting codes, viz. that for fixed burst length, the Hamming bound gives asymptotically the minimum achievable redundancy. (This work has been done jointly with Andrew Odlyzko of AT&T Bell Labs.)

We are preparing several papers for external publication to report our findings. A preliminary version of the paper "two-dimensional burst identification codes" was presented by Robert McEliece at the 1985 IEEE workshop on Information theory, held at Caesaria, Israel, last summer; and a preliminary version of the work on one- and two- dimensional cyclic burst error correction will be given by McEliece in April 1985 at the 10th Carribean Conference on Combinatorics and Computers.

4.2 CAD Tools

Steve Rabin

Miscellaneous problems with the IC verification tools have been uncovered and fixed. The artwork circuit extracter has been upgraded to run under version 8 Mainsail, ported to several our new machines, and optimized to run 30% faster. Our design tools are now capable of verifying bulk CMOS circuit designs.

4.3 Mossim Simulation Engine

Bill Dally

Design verification is essential in the development of VLSI systems. The complexity of VLSI circuits, inaccessability of internal nodes, and difficulty of repair make the probability of producing a working chip very low without extensive design verification. As the complexity of VLSI circuits approaches 10^6 devices, the computational requirements of design verification are exceeding the capacity of general purpose computers. To provide the computing power required to verify these complex VLSI chips, we are developing the Mossim Simulation Engine (MSE) [5123:TR:84].

The Mossim Simulation Engine is a special purpose processor which, in a single processor configuration, performs switch-level simulation of MOS VLSI circuits 200-500 times faster than a VAX 11/780. In multiple processor configurations even greater speedup can be achieved.

The MSE overcomes two limitations of existing simulation engines. By using the switch-level model developed by Bryant [5065:TR:83], the MSE performs accurate simulation of MOS circuits. Existing simulation engines perform logic simulation and cannot model MOS effects such as stored charge, charge sharing and transistor ratios. Also, by using the concept of virtual processors the MSE can simulate a circuit many times larger than the size of the processor. Existing simulation engines are limited to simulating circuits which fit in the processor.

A prototype MSE has been constructed and is now working. Initial experiments on the prototype machine show a measured performance of 150K gate evaluations per second only slightly less than the performance estimated by simulation. The only MSE software currently available is a hardware debug monitor. Systems software: a run time system and a user interface, to make the MSE available to VLSI designers is being planned.

The prototype will be used both as a research tool to support the simulation requirements of the next generation of VLSI circuits and as a test bed for experiments in switch-level simulation including: the development of a virtual simulation processor system, experiments in the application of multi-processing to switch-level simulation, and a study of the locality of activity in MOS circuits.

A paper describing the MSE is attached.