SUBMICRON SYSTEMS ARCHITECTURE

28 October 1982

SEMIANNUAL TECHNICAL REPORT

5052:TR:82

Edited and in part written by

Lennart Johnsson and Charles L. Seitz

Co-principal investigators: Charles L. Seitz, Carver A. Mead, Lennart Johnsson

Other faculty: Randal Bryant, Jim Kajiya, Alain Martin, Martin Rem

Staff: Jim Campell, Vivian Davies, Pete Hunter, Michael Newton

Ph.D. students: Young-il Choo, Erik DeBenedictis, Peggy Li, Sheue-Ling Lien, Mike Ullner, Dan Whelan, Doug Whiting

M.S. students: Bill Athas, Chao-Lin Chiang, Howard Derby, Eric Holstege, Chris Lutz, Charles Ng, John Ngai, Craig Steele

Undergradute students: Steve Rabin, Don Speck

Computer Science

CALIFORNIA INSTITUTE of TECHNOLOGY

Table of Contents

1. Overview

1.1 Scope of this Report

This document reports the research activities and results for the period October 1 1981 - October 15 1982 under the Defense Advanced Research Project Agency (ARPA) Submicron Systems Architecture Project.

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 design tools.

Much of the work at Caltech on advanced design tools and designer workstations is supported separately within the industry- and NSF-sponsored Silicon Structures Project (SSP), and is reported elsewhere. Where there is some overlap in support between ARPA and SSP, or where a graduate student may be supported by a fellowship, this additional support is indicated in reporting that effort.

1.2 Scope of the Research

The submicron systems architecture project is a carefully integrated research effort, in which theory is closely tied to experiments in VLSI architecture and design. The various aspects of the research tend to support each other. To the extent that they can be separated, the several research areas can be listed as follows:

- **Architectures** that can exploit the capabilities of VLSI technology.
- **Concurrent algorithms** for important and demanding computations.
- Theory of **Concurrent computation** applied to the representation of algorithms and switching systems.
- **Design disciplines** that manage the complexity of VLSI systems.
- **Design tools** that check and enforce these disciplines.

Let us try to explain the close connection and common motivation of these areas.

What is driving this research effort are the opportunities and problems presented by VLSI technology. As microcircuit technology is scaled to features at submicron dimensions, it presents an opportunity of a dramatic reduction in cost/performance, represented by the cube-law scaling of the switching energy Esw, and of an increase in the complexity of chips. However, there are attendant problems in learning how to exploit this opportunity.

Where communication is recognized even today as being expensive and limiting in chip designs relative to earlier technologies, scaling makes this problem worse. In order to be able to exploit the technology to achieve performance gains, it is necessary to localize communication and to free the system parts to operate concurrently. The VLSI architectures that appear most promising to us are concurrent systems,

and require for their effective use programming styles and algorithms that exploit concurrency and localize communication.

Managing the complexity of VLSI systems, both in numbers of elementary parts and in the many levels of abstraction employed in a design, requires a deliberate and rigorous structuring of the design process. Design disciplines -- or less formally: design styles, or methods -- are the theoretical foundations of practical design, and are both suggested and tested by adhering to these disciplines in designs and design tools.

1.3 Research Summaries

The following is a summary of the research described in more detail in sections 2 through 6 of this report, and which is in turn described in complete detail in the technical reports cited and listed in section 7.

ARCHITECTURAL EXPERIMENTS

Currently, there are two architectural experiments involving constructing working systems under way at Caltech: the Tree Machine Project, and the Homogeneous Machine Project. Each involves both hardware and software efforts.

The Tree Machine is an ensemble of deliberately very small processors, each with a small amount of storage, and each with either 1 or 3 communication channels to allow message passing in a binary tree communication plan. The scale of the individual element of the ensemble is here so small that such a system built with 3-4 micron nMOS technology allows 2 to 4 elements per chip, and for submicron technology would allow 2^7 or 2^8 elements per chip.

The node processor for the Tree Machine is of our design, and incorporates several other design experiments. The first iteration on the design is just now approaching completion. It is a small (2M lambda^2) 16-bit processor, projected by simulation to be quite fast. The structured design methodology of [Mead, Conway 80] is used, but the electrical and physical design rules used as well as use of the clocks are different. We hope to establish a new set of electrical design rules as one of the results of this experiment.

The Tree Machine project also serves to verify that the model of synchronization of processors via message passing can be efficiently implemented. The communication protocol is built into the communication channels of the Tree Machine processor.

Eventually, experience will also be gained from constructing a tree machine with enough processors (e.g. 1023) that it achieves considerable computational power. Software tools for writing, assembling, loading, execution, and debugging of programs written in a notation with restricted communication primitives are already in place and used with a tree machine simulator.

The Homogeneous Machine is an ensemble of much larger processors, each with sufficient storage to contain a run-time system for spawning and garbage collecting processes in execution, storage management, routing messages, and language support. The Homogeneous Machine also employ strictly message passing communication, with no shared storage, and the

communication plan employed in current experiments is a Boolean n-cube (hypercube) connecting ensembles of 2^n processors using n communication channels per processor. Where with today's technology it requires in the order of 100 chips per element, this experiment is aimed at a size of processing element that at submicron dimensions scales to 1 or 2 chips.

A 6-cube (64 processor) machine is currently under construction and should be in use by the end of the calendar year, and a 2-cube (4 processor) prototype is running concurrent programs. A 10-cube (1024 processor) machine is being planned.

This experiment serves to gain additional experience in efficient interprocessor communication, in distributed operating systems, and in the implementation and use of high level concurrent languages. Several programming experiments are underway, including the application of this machine by collaborators in high-energy physics to scientific computations that require high performance in matrix, differential equation, and distant field computations.

DESIGN DISCIPLINES

Our objective of producing one or two exemplary and full scale self-timed designs has been well accomplished by a family of communication chips that support either one-to-many self-timed communication over separable bus structures, or one-to-one self-timed communication on bidirectional serial channels. This delay-insensitive communication is accomplished between locally synchronous systems, such as an ensemble of microprocessors. At another level this experiment has been aimed at discovering a VLSI counterpart to the macromodules developed at the Washington University Computer Systems Laboratory a decade ago. Like macromodules, these communication chips allow an "electrically naive" person to assemble an arbitrarily large system without getting into timing difficulties. However, where macromodules are assemblies of registers, operators, storage, and control elements, these communication chips allow one to assemble large systems whose basic parts are microprocessor-based computers.

The design methods we have been experimenting with in the VLSI design laboratory class over the past year have been focused on developing simplified design rules for CMOS/SOS following the style of the Mead & Conway rules for nMOS, and on design for performance techniques for nMOS. Design experience in CMOS/SOS has been accumulating through student projects. Fabrication services for 5 micron CMOS/SOS for the past year were arranged through Asea-Hafo in Sweden under a no-cost arrangement, and enough test structures and projects have been checked to confirm the basic design style. A new set of geometrical design rules and electrical parameters for CMOS/SOS, based on an RCA 4 micron process, has been developed and made available to the ARPA community through MOSIS. A revised (to the new design rules) CMOS/SOS library and design guide is in preparation.

In the area of testing, the definition and implementation of an interactive test language has been completed. This language is based on an abstractive model and mechanism for merging testing and hierarchically structured design, and the testability strategies

suggested by this language and model have been analyzed to determine the expected complexity of test sequences. Because the language is based on an hierarchical model, tests structured in this way can also reduce the amount of data transfer between a host computer and a tester.

The test language is implemented by the FIFI test system. The FIFI test system has two parts: a small experimental tester, and a (software) test language interpreter.

DESIGN TOOLS AND WORKSTATIONS

Earl and MOSSIM are the principal design tools whose development and maintenance is supported under this contract, as well as those currently most heavily used in designs.

Earl is a programming language and associated interpreter for specifying connectivity and geometry. Earl is based on a separated hierarchy model for geometry, in which geometrical primitives such as wires and transistors are specified strictly within a cell, and the composition of cells to produce larger cells is specified by composition operators that work by abutment. Earl's interpreter includes constraint solution for determining the location of connection points, or other declared points within a cell design, and adjust the cell geometry including stretching the cell according to the way the cell is composed with other cells.

MOSSIM II is a switch-level simulator that originally was developed at MIT. It has been modified and further developed by its creator, Randy Bryant, now at Caltech. The current version, MOSSIM II, is written in Mainsail. It is available along with a set of programs for hierarchical network specification, circuit extraction, and network comparison. MOSSIM is based on a highly evolved model of switch logic in which nodes may be determined in simulation to be logical 0 or 1, or undefined (X). Different modes of simulation can be set to correspond to different degrees of conservative choice about wire and switch delay. The models employed by MOSSIM have accordingly been of interest also in investigations of synchronous logic disciplines, speed-independence in self-timed systems, and most recently in test pattern generation.

A new effort in design tools is underway to build a tool that can be used at the final assembly stages of a chip design to route together cells produced by tools oriented to geometry rather than interconnection, either Earl or graphics-based tools. This effort is based on a hierarchical model of chip assembly with progressive use of additional metal layers.

In the development of a designer workstation the focus has been on a fast display system. The idea is that a layer in a given representation be represented by a bit plane, and that the different planes be updated concurrently. A prototype system based on NEC's graphics processor has been running for several months. Even though the architecture of the NEC processor limits the performance of the prototype system, it is useful and the experiment has been encouraging.

CONCURRENT ALGORITHMS

As the feature size of the technology continues to decrease and the design and implementation tools evolve, the design of custom circuits of significant complexity should become routine. A great proliferation in architectures can be expected. The physical properties of the implementation medium calls for concurrency in operation for its efficient use. The efficiency of architectures for concurrent computation depends largely on limiting and matching communication patterns of the algorithm with that of the implementation medium, be it a chip or programmable concurrent machine. By mapping algorithms in "space", the sequencing enforced by a single processor is avoided in favor of concurrency.

Most of our studies of architectures for submicron technology focus on specific computationally demanding applications, and for which the highest performance machines are relatively special purpose, but "general" computing systems are considered as well. So, the main research areas are somewhat divided by the implementation target:

- Architectures supporting concurrent programming at various levels, and which can be used in addition for the applications below, but at less than optimal performance.

- Architectures directly structured for concurrent algorithms with application to scientific computing, signal and image processing, and computer graphics.

The Tree Machine and Homogeneous Machine projects serve as vehicles for studying architectures for concurrent programming languages and for scientific computing. For the latter aspect close ties have been established with the Applied Mathematics and High-energy Physics groups at Caltech.

A set of concurrent algorithms for the standard computations in linear algebra have been devised. The common characteristic of these algorithms is that they can all be implemented in computational arrays made up of processors containing a few registers, an arithmetic unit, and very limited program store. The arrays are essentially SIMD machines, even though not all processors always perform the same instruction. An array can be made to perform a set of operations such as matrix-vector, matrix-matrix multiplication and solving linear systems of equations simply by routing the data in different ways through the array. Program storage is only required in the control unit. It should be noticed that the programmability is bound at design time to a very limited set of operations.

The study of concurrent architectures for image processing has taken fingerprint analysis as a sample application. Most of the effort has been devoted to finding suitable encoding of fingerprint images and concurrent algorithms for their analysis. A fingerprint image (ridges and forks) is first encoded as a graph whereupon a comparative analysis is made. The first step is performed iteratively in a serial raster scan fashion using nearest neighbor operations. The core of operations for the second phase deals with graph isomorphisms.

Ray tracing techniques have the capability to produce some of the most realistic synthetic images. Ray tracing is often considered intractable because of the amount of computation required. However, concurrent algorithms for ray tracing techniques can be devised. The processing elements can be kept simple and the amount of concurrency enormous. In the past year concurrent implementations of ray tracing algorithms have been investigated and new algorithms been devised. Machine architectures have also been conceived for some algorithms that holds promise to offer real-time performance. One such machine produces shaded images of a scene using scan line techniques; another eliminates hidden lines from a wire frame picture.

A display system that has rectangular area filling as a primitive has been designed. Filled areas are rendered in constant time, i.e., an $O(N^2)$ speed improvement over conventional pixel based displays. The performance improvement for filling convex polygons is O(N). A chip performing the area filling function has been designed, the first few were returned from fabrication in early October, and they are currently being tested.

FORMAL TREATMENT OF CONCURRENT SYSTEMS

The need for a formal treatment of concurrent algorithms and programming models is far greater than for sequential systems, and is far more difficult. Research in this area is still in an early stage, and there is not yet a commonly held view of which models are "good" or even correct at various levels of system representation. Models of a system are often postulated at one level of abstraction without any assurance that the model is consistent with lower level models and the technology. In this research area alternative, even contradictory, approaches are tolerated.

Two notations for the description of concurrent algorithms are being explored. One has been strongly influenced by C. A. Hoare's CSP. A semantic definition of the communication primitives has been proposed. Channel variables are introduced and input commands in guards are disallowed. The semantics of the communication primitives used is captured by two axioms. A synchronization axiom formalizes the synchronization properties of the primitives, and a communication axiom formalizes the distributed assignment property of the primitives.

In CRYSTAL (Concurrent Representation of Your Space-Time ALgorithm) the semantics is based on the fixed-point approach. A program is expressed as a set of recursion equations. For a deterministic concurrent system a single system of equations results. The semantics of such a system is defined as the least solution of the equations. The semantics of general concurrent systems is defined as the set of solutions of the set of system equations. CRYSTAL has been applied to: transistor circuits, gates, arithmetic units, sequential processors, large ensembles of communicating processors such as systolic arrays performing matrix multiplication, and tree machine algorithms. Both synchronous and self-timed communication have been modeled.

Synchronization in concurrent system has been studied with the following result:

- Two algorithms have been devised and verified to accomplish distributed mutual exclusion: one applies for processors interconnected as a ring, the other for processors interconnected as a general graph.

- A theorem characterizing necessary and sufficient conditions for freedom from deadlock in an ensemble of processors has been proved.

- A structured Petri net model, called Hierarchical Nets, has been devised and studied.

Finally, a few experiments in developing functional or logic programming languages are in progress. Fith (with several ideas from FORTH) is an extremely small language which explores the applicability of data abstraction and late binding to systems programming. ATP (inspired by APL) attempts to include data abstraction in a functional programming language. '81 attempts to fuse functional and logic programming.

2. Architectural Experiments

Chuck Seitz and Lennart Johnsson

The VLSI architectures that have been investigated at Caltech over the longest period, and are accordingly at the stage where we may report them as experimental machines at least under construction, are systems that are ensembles of identical, concurrently operating, and regularly interconnected elements. We refer to these systems as **ensemble architectures.**

Ensemble machines may be classified by a crude taxonomy [Seitz 82a] according to the size of the element that is replicated. The generality of the system is determined largely by this element size, the performance in the best case is linear with the number of elements, and the cost varies as the product of the element size and number of elements. At fixed cost, then, one encounters an inevitable competition between generality and performance. It should be stated that our priority and short-term objective in this research is to apply VLSI technology to achieve a substantial advance in cost/performance in computationally demanding tasks, rather than to try to achieve generality. This greed for speed has at least the advantage that cost and speed can be measured, and such "benchmarking" is an integral part and measure of each effort.

Our earliest (1977) research on VLSI architectures at Caltech started with very fine grain ensembles sometimes referred to as "smart memories," and through the years the emphasis has evolved towards the present research focus on three classes of ensemble machines with increasingly larger element size:

- Computational Arrays, also called systolic arrays, although they need not be synchronous. Typical element size is 0.1 to 1 million square lambda, with the element capable of performing arithmetic operations such as multiplication and addition on operands in internal registers or communication ports. The applications that have been studied include matrix, including sparse matrix, and signal processing computations. Although these are wired algorithm machines, the element will generally be parameterized or even microprogrammable. Although we have not yet identified a computational array for an experimental implementation, the study of algorithms for this class of machine, taken as a computational model, has been vital to the programming efforts of the two more general classes, and is described in detail in section 5 of this report.

- The Tree Machine, or other programmable ensembles with a very small processor and associated program and data storage per node. Typical element size is 1 to 10 million square lambda. It is difficult to build an even marginally programmable machine with element size below 1 million square lambda, and the machines in this class are close enough to this limit that the programming techniques are fairly low-level. The applications that have been studied include those of less general ensembles, as well as several graph and combinatorial problems. The tree machine element that

has been designed and is described below is about 4 million square lambda, and so can be fabricated even in today's technology with more than one element per chip. Although the tree interconnection is too restrictive for some problems, it has the advantage of allowing more processors per chip with a fixed number of pads.

- Homogeneous Machines are ensembles of processors that are individually powerful enough to contain extensive run-time systems, and typical element size ranges upward from 10 million square lambda. Although the applications of these machines include all of those of the less general classes, the study of applications here focuses on programming methods that allow one to break out of the rigid communication planning of the less general classes. It has been a somewhat surprising result for us that even in building a small version of such a machine from catalog parts, the cost/performance gains over mainframe uniprocessors is substantial, factors approaching 100 for the regular problems actually benchmarked on this machine. The 64-processor Boolean 6-cube machine being constructed (a 4-processor 2-cube prototype is being used for program development and is running concurrent programs) requires about 100 million square lambda (77 chips), and could be expected to scale to 1 or 2 chips per element at 0.5 micron feature size. In anticipating that multiple elements per chip is infeasible in the scaling of this class, and that relatively high communication bandwidth is desirable with less than regular problems, a relatively richer but always wirable communication plan of a Boolean n-cube has been chosen for current efforts.

What makes all of these structures such excellent candidates as VLSI architectures is that (1) replication is exploited extensively, and (2) the cost/performance can be projected to track with the cube-law scaling of Esw (the switching energy) as feature size is reduced to submicron dimensions. These structures also have in common a complete avoidance of shared storage and its attendant scaling difficulties in favor of message passing.

The problem that we see as being addressed in these efforts that is not addressed by commercial developments in micromainframes is performance. The fundamental problem is that the scaling of Esw is the product of a quadratic scaling of area and power per function and an only linear scaling of transit time. It is accordingly easier to exploit advances in the microcircuit technology to reduce cost (power and area) than to enhance performance. An analysis of scaling whole designs, including their wires [Seitz 3863:DF:80], [Mead, Rem 81], indicates that there is a point of diminishing returns in large designs where the scaling of transit time is not reflected in system performance [Seitz 82a]. We are accordingly forced to use concurrent machines, whether we want to or not.

While the scope of the entire project includes several investigations of programming models and languages for concurrent computation, there is no pretense that the programmable classes of these experimental ensemble machines can today be programmed as easily as conventional computers, nor can they be applied efficiently to more than a narrow range of regular problems.

Indeed, even using the best notational and theoretical tools at our disposal, the design of efficient algorithms for and the programming of these machines is difficult. However, the long list of regular and almost regular problems that are now fairly well understood includes many of the computationally demanding tasks in physics, geophysics, chemistry, and applied mathematics. The strong interest of our colleagues in these particular disciplines at Caltech, and their eagerness to invest their own research time and funds in rethinking their algorithms to apply these machines to their own computationally difficult problems, is some indication of the need for performance and the range of application of these machines.

2.1 The Tree Machine Project

2.1.1 Background:

Chuck Seitz and Lennart Johnsson

The Tree Machine is an ensemble of concurrently operating, small, and marginally programmable processors, each with its own storage, and interconnected in a binary tree. This interconnection allows a processor to send messages to its parent or to either of its descendants.

Sally Browning's studies [Browning 3760:TR:80] of algorithms for and the capabilities of this machine, completed in January 1980, was the starting point for this project. In addition to being an interesting structure in its own right, the regularity and testability characteristics of this machine made it attractive to continue this architectural experiment to make a working machine as a test vehicle for building a large system based on chips of our own design.

After working out some details of the interprocessor communication [Browning, Seitz 81], a small 12-bit processor was designed with a 4-bit path to program and data storage that was to be implemented on a separate chip. Efforts in developing a software system, led by Peggy Li, proceeded from this point concurrently with efforts in designing a processor chip.

Two attempts to layout this processor were unsuccessful due to weaknesses in design tools. The first layout attempt, by Chris Lutz, Howard Derby, and Chris Kingsley, over the summer 1980, proved to exceed the address and name space capabilities of a Simula-based extension to LAP, but was the origin of the new design tool Earl [Kingsley 5021:TR:82], written during 1981.

A second attempt, by Peggy Li, to lay out the 12-bit tree machine processor with more advanced tools also failed due to incompatible design tools that were also unable to handle a design of this size. The data path was specified using the language of Bristle Blocks. The control part was designed as five interconnected PLA's, whose microcode was written in ICLIC. This second design effort was given up in mid April, 1982.

Finally, however, we have succeeded in designing a Tree Machine processor node, one that is more ambitious than the original design, and the project has clearly been an excellent stimulus and reality test for our design capabilities and tools.

This project was conceived as an experiment with a regular interconnection of nodes that are about as small as reasonably could be programmed, in a tree structure that is easily wirable independent of the number of nodes per chip, and which has good testability characteristics. Once this chip is developed into a robust and manufacturable design, what kind of architectural experiments does it suggest?

As reported below, the layout of this processor is only 1400 by 1400 lambda, or 2 million square lambda, and could still be improved somewhat. It should also be quite fast, much less than a 200 nsec clock period according to SPICE simulations. The (3-transistor) dynamic RAM array designed for it, with test chips sent to MOSIS in August, has a cell size of about 250 square lambda per bit, plus typically 20% as much additional space used for address logic and drivers. If a similar area, about 2 million square lambda, is used for storage as for the processor, the total storage would be around 400 16-bit words. Fixed subroutines, loader, and other "system" software can be kept in ROM, which being about 4 times denser would result in a system having about 256 words of RAM, and about 512 words of ROM. This is quite a lot more storage than originally anticipated, and about 4 times more than any of Sally Browning's programs require.

At 2 micron lambda the processor is only 2.8 mm on a side. This small size means we can fit 2 processors with storage of similar size on a single chip at 4 micron feature size, or

feature size (microns)	4	2.8	2	1.4	1	0.7	0.5
# of processors/chip	2	4	8	16	32	64	128

with a constant 6 mm on a side chip. So, whatever feature size fabrication we can get our hands on, we can use. A 2K chip system at 2 micron feature size would be a 16K processor, or 14 level tree, which represents a lot of performance for special-purpose applications in a small box. Such a machine could, for example, invert a 100 by 100 matrix of floating point numbers in less than a second.

The Tree Machine processor includes a (LSSD) scan path in the design between the control PLA and the datapath. This feature would be retained even in chips with many processors, but not strung together serially as one might first expect. The chip LSSD input would go to the LSSD input of all the processors in parallel. The chip LSSD output would come from the LSSD output of one processor, and there would be a second chip LSSD output that would indicate whether any other processor's LSSD output disagreed with the first LSSD output. In this way one can perform the initial test of all of the processors in parallel, and this segment of the testing time is constant in the number of processors per chip (or system).

The second phase of testing, performed after correct operation of the processor control and sequencing is assured, is to test the interprocessor communication paths. This test easily fits in the storage of each processor, and has a time complexity of the log of the number of processors, with a fairly small coefficient.

The third and final phase of testing -- storage, registers, arithmetic, and other datapath operations -- is most efficiently accomplished by downloading programs that run in all the processors at once. This test also has a constant time complexity independent of the number of processors.

So, we have here a case of an architecture whose testing scales so that it is essentially constant between a chip with one processor and chips with hundreds or thousands of processors and millions of bits of storage.

2.1.2 Tree Machine Processor Design:

Chris Lutz (Advisor: Chuck Seitz)

A new Tree Machine processor was designed during the period from January to June 1982 [Lutz 5036:DF:82]. With smaller lambda than in 1980 and with buried contacts now available, one could put both processor and storage on a single chip, so many of the arguments in favor of a narrow path to primary storage were obsolete. The new machine was accordingly designed with a 16- rather than 4-bit path to storage, 16- rather than 12-bit words, and an instruction set similar to the original design but based on a 16-bit word rather than a variable number of 4-bit nibbles. The basic communication scheme from the original design was also preserved.

The data path includes, working left to right across the floorplan: up to 4 serial output ports, up to 4 serial input ports, 16 registers, operand latches, function blocks, ALU, shifter, and address and flag logic. There are eight addressing modes (the combination of source/indirect, immediate/value and destination/indirect), and a 12-bit address.

The control is microprogrammed and pipelined. Storage cycles are completely overlapped with microprogram cycles and datapath operations. At least one clock cycle in every instruction that would be otherwise unused for a storage reference is used to refresh the dynamic storage with a refresh address register internal to the processor. The microcode also includes a very simple program loading initialization bootstrap. An example of the degree of pipelining in the control is an instruction that adds to register A the contents of the storage location pointed to by register B and leaves the result in that storage location. This instruction requires 5 clock or microcode cycles; two of these cycles are overlapped with normal storage cycles and one is overlapped with a storage refresh cycle and with an ALU operation.

Even though the instruction set is fairly rich and the control complex, the microprogram PLA requires only 96 terms (implicants), folded into 48 rows in the PLA in order to better match the floorplan and PLA cell layouts. The processor was simulated functionally at the microcode

level before layout was started, and more recently is been simulated at the switch level with MOSSIM.

As indicated above, an LSSD scan path is included between the microcode PLA and the data path, and allows the microcode contents and control of the datapath to be tested efficiently.

2.1.3 Layout:

Chris Lutz, Don Speck, Steve Rabin, Pete Hunter (Advisor: Chuck Seitz)

A major effort to lay out the new design of the Tree Machine processor was undertaken between July and October 1982. The area for this complete processor, including 4 input and output channels, but not including the I/O pads, is only about 2 million square lambda, and the clock period projected from SPICE simulations is less than 200 nsec. The design was done by a team of 5 designers, and has required a total of approximately 10 person-months so far.

Continued efforts in this design are concentrating on verification and simulation in preparation for the November MOSIS nMOS run. Thus far the simulations have revealed only simple and easily corrected errors in the logic design and microcode. A future version of the layout will replace the present static microcode PLA with a precharge PLA.

Two dynamic RAM test chip designs to provide on-chip storage for the tree machine processor were completed and submitted to MOSIS for fabrication in August.

A standard test strip for checking the tolerances of the bootstrap driver circuits to process variations was designed and has been submitted for all MOSIS nMOS runs since June 1982.

All designs are being done using Earl, a geometry tool originally developed in connection with this project. Earl is described in more detail in section 4 of this report. Other tools used in these designs are SPICE, MOSSIM, an extractor, and a program to compare extracted layouts with MOSSIM.ndl files.

The Tree Machine processor is also being used as a vehicle to develop for nMOS technology a high performance design style in which clocks switching between ground and a voltage in excess of VDD are driven directly onto the chip, and supply much of the power required to operate the chip. These "hot clock" signals are processed by bootstrap clock-and circuits that preserve the clock voltage, effectively only steering the clock signal to different control lines. This technique, which is used both in the Tree Machine processor and in the address decoding in its companion 3-transistor dynamic storage, is very appropriate to the Tree Machine architecture, in that a single high-performance (bipolar) clock driver can be shared by many nodes.

At least according to SPICE circuit simulations, this "hot clock" technique has exceeded our most optimistic expectations both in performance and in area savings. It is suggestive of a style that overcomes some of the deficiencies of nMOS, particularly in scaled form. The Tree Machine processor is designed for VDD = 5 volts and clocks switching to 7 volts, and a 2 to 4 micron process. Clocked pass gate

structures do not produce degraded output signals, so full output swings are maintained and normal 4:1 ratios may be used. The pass gate structures in the ALU carry path do not produced degraded output signals because the path is precharged. With a 1 micron nMOS process, the VDD would be reduced to about 2.5 volts, which with ordinary design styles would require tighter tolerances on many of the circuit parameters. However, with a higher voltage direct clock switching between 0 and 4 volts, the tolerances required are not substantially tighter than those in force today.

2.1.4 Software:

Pey-yun Peggy Li (Advisor: Lennart Johnsson)

(Currently also partially supported by an IBM fellowship.)

2.1.4.1 Background

A simulator for the Tree Machine has been in existence for several months. It has been modified as the instruction set has been refined in the simulations and layout of the processor.

The tree machine assembly language, [Li 4618:TM:81], contains notations to define both the connection plan of the tree and the program of each node. The programmer has the freedom to create a tree with arbitrary fanout and arbitrary size. The assembler converts the logical tree into a binary tree.

The three pass assembler performs the following tasks:

- convert the logical tree into a binary tree by inserting padding nodes into proper positions,
- generate code for padding nodes so that they can serve as communication channels between a node and its descendants,
- macro expansion and macro definition,
- generate the machine code for each node,
- provide the necessary information for the downloader.

The downloader is written in the Tree Machine assembly language. An interactive user command interface was built up on top of the simulator. The first version of the Tree Machine operating system was running in June, 1981.

Over the past year, the major effort has been concentrated on the following improvements:

2.1.4.2 Upgrading the simulator:

The simulator has been upgraded in the following sense

- Add a virtual memory feature to the local memory of each processor.

- Simplify the data structure of the simulator.

The purpose is to save the state and reduce the simulation time.

2.1.4.3 New downloading algorithms

The original downloading algorithm has a time complexity of O(N), where N is the total number of nodes in the binary tree. A complexity of this order is undesirable, since the downloading time increases exponentially with the height of the tree. The downloader loads two files into the tree, one is the module name file, which stores the name of each node in the binary tree, the other is the module code file, which stores the program code for all the different modules in the tree. The size of the module name file is proportional to the total number of nodes of the binary tree and it becomes the critical part as the tree grows.

In our first attempt to reduce the size of the module name file, identical nodes occurred only once in the module name file. The number of occurrences was specified by an integer preceding the name. Parentheses were used to allow for the definition of subtrees. This scheme reduces the size of the name file to be proportional to the number of different nodes in the tree. For a very regular tree the name file tends to be proportional to the height of the binary tree, a quite satisfactory result. However, if the logical tree has an odd fanout, the converted binary tree is very irregular and the size of the name file can not be satisfactorily reduced. Another problem for this scheme is that the format of the module name string is not unique. To find the shortest string is an NP-complete problem.

In our second attempt the structure of the logical tree is given directly. The loader in each node generates the padding nodes. Each module is represented by a triple (the number of successive identical nodes in each logic level, the name, and the fanout). With this second scheme the size of the name file is three times the number of sets of nodes that need to be loaded. The number of sets is at best equal to the height of the logic tree, at worst equal to the total number of nodes in the logic tree. The total downloading time is proportional to the sum of the length of the name file and the height of the binary tree.

2.1.4.4 New version of the assembler and the simulator

All the Tree Machine software, including the assembly language, the assembler, and the simulator have been modified according to the instruction set and the architecture for the current 16-bit machine. The downloader and the padding procedures have been rewritten in the new assembly language. A single processor option has been added into both the assembler and the simulator, so that the assembler now can generate the code for any specified processor in a tree and the simulator can simulate the same processor. New commands have been added into the simulator to save and restore the memory image of the Tree Machine into/from a virtual memory file. The user need not waste time to load

the code into the tree every time.

2.2 Homogeneous Machine Project

2.2.1 Communication Modeling of Homogeneous Machines

Dick Lang (Advisor, and reported by: Chuck Seitz)

The PhD thesis completed by Dick Lang in June 1982, "The Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture," [Lang 5014:TR:82] represents a substantial step in understanding both the engineering and programming of homogeneous machines. As indicated by the references below, the benchmarking by simulation that was accomplished has provided a useful guide to many of the engineering considerations for these machines, including interconnection topology, necessary bit rate, packet size, and routing algorithms.

The programming model developed for its own sake and used for these simulations is an idea that was "in the wind," namely, to use the class-object programming paradigm as an approach to localizing data with the programs that operate upon it, and to treat the act of invoking an attribute of an object as a communication between spatially distributed objects rather than as only a co-routine type of mechanism. The semantics of Simula are reinterpreted so that execution of a statement such as "vec.sort;" sends a message to the object "vec" to sort itself. Continued execution in the sending object, when no immediate result is expected, is the basic mechanism for concurrency.

The essential constraint on the objects and message passing system is that objects (unlike actors) are sequentialized, and therefore serve also as critical regions, and that successive messages sent from a particular source object arrive at the destination object in the order in which they are sent. It is necessary also, of course, to eliminate global variables. Some of the programming examples in this notation are both valid sequential programs and valid concurrent programs.

If producer-consumer problems are to be represented, it is necessary to include an extension to Simula referred to as a SELECT, and which allows an object to determine from separate queues for each attribute which attribute may next be executed, a function similar to the guarded command in CSP.

The essential global process required in such a system is storage management, including particularly garbage collection. A relatively brute-force algorithm requiring a single loose synchronization (supported in the hardware) was devised, verified, and simulated, with the result that garbage collection on this distributed system requires only a small fraction of the system performance, even in very large homogeneous machines.

The simulation of several connection plans, chordal ring, tree, mesh, and Boolean n-cube (with 3 different routing algorithms) revealed the expected result that the most expensive of these connection plans, the Boolean n-cube, was substantially superior at least for the models of message locality derived from assumed object distributions ranging from random to quite well optimized. Some of the results for meshes were inconclusive due to deadlock, lacking any knowledge at the time of a suitable deadlock avoidance routing algorithm. A very simple routing algorithm is known for the n-cube, and is assured to avoid deadlock, and the simulated performance of the system using this routing algorithm is typically 50% to 70% that of higher performance but unsafe routing algorithms.

Finally, this study revealed a great many problems that could not be adequately studied by simulation, particularly questions concerned with maintaining locality among object references, relocating objects from congested areas, and locating previously relocated objects in a machine. Questions of the performance of systems using these more ambitious techniques will have to be answered by system building experiments on a real homogeneous machine.

2.2.2 The 6-cube Homogeneous Machine

Erik DeBenedictis, Michael Newton (Advisor: Chuck Seitz)

A 4-processor (Boolean 2-cube) prototype of the 6-cube machine under construction is now running concurrent programs. The node processor is an Intel 8086 microprocessor with 8087 floating point chip, 128K bytes of storage, and 6 queued and asynchronous bidirectional channels. The machine is complete including the dedicated host and interfaces to other department machines and to the high energy physics VAX.

Benchmarks for partial differential equation solution run on this 5 MHz 4-processor machine at 0.67 of a VAX11/780, confirming the benchmarks for the single processor previously reported of 1/6 of a VAX11/780 per node for 5 MHz operation, or 1/3 of a VAX11/780 for 10 MHz operation (which awaits availability of 10 MHz 8087 parts).

Given this verification of the design, construction of the 64 processor Boolean 6-cube is proceeding. A printed circuit board for the node processor was layed out, prototype boards were constructed, and a printed circuit board version of the node has been assembled and is being checked out. None of the preliminary tests of the node have revealed any problems, nor would any be expected since the net list for routing the PCB was derived from the wirewrap net list.

Circuit and packaging parts are on order to complete the 6-cube machine by the end of the year.

The resulting machine is projected to have 20 times the performance of a VAX11/780 when running at 10 MHz, and twice the main storage, at about 1/5 of the cost, representing an improvement in cost/performance for those problems for which it is suited of a full 2 orders of magnitude. This entire machine will be 14.2 by 8.5 by 63 inches in size, and consume 700 watts.

Programs for the 6-cube are being developed and tested on the 2-cube

both by a group in high energy physics and by groups in computer science. Most program development is depending on the C compiler developed by high energy physics, and on normal Intel 8086/8087 assembly tools.

The physics group is concentrating on extremely regular matrix, differential equation, and grid point problems, using a programming style that we have come to refer to as "crystalline." Much of the effort in computer science is currently centered on system software development, and less regular applications such as combinatorial problems.

2.2.3 The 10-cube Homogeneous Machine

Chuck Seitz

Discussions with several computer manufacturers have stimulated interest in a joint project to design and build a 1024-processor Boolean 10-cube machine, with the computer manufacturer supporting the project by contributing the hardware. This approach appears to be an ideal way to carry the concept forward, since it (1) allows continued software and applications development in the university environment where it has historically thrived, (2) places the logistic problems of building such a machine in an industrial environment that is well organized for the task, and (3) makes machines of this type commercially available, unlike experiments such as the ILLIAC series, CM*, and others, that are or were unique machines.

A 10-cube machine produced with a node processor based on one particular manufacturer's 2-chip minicomputer would have a performance for regular concurrent problems of several thousand Mips (million instructions per second), or well in excess of 10 times the performance of the most powerful general-purpose machines of today, and would have an (internal) cost of about $1M.

In anticipation of the design of a larger homogeneous machine, a communications chip specialized for serial communication in a Boolean n-cube has been designed, and a first version submitted to MOSIS for fabrication in August. This chip is a variant on an earlier design previously reported of a self-timed chip set for multiprocessor communication (see section 3).

The Homogeneous Machine communication chip includes 32 words by 16 bits of FIFO in each direction, and implements on a single chip a highly efficient delay-insensitive serial communication scheme. About half of the board area of each node of the 6-cube machine is required to implement its 6 communication channels in small- and medium-scale integration chips. This custom chip replaces all of this "glue," and so represents a substantial savings at each node, as well as increased FIFO storage.

2.2.4 Programming notations

2.2.4.1 General

The performance of homogeneous machines depends on the separability of the problem into many concurrent parts. Programming homogeneous machines depends on the ability to express concurrent algorithms in a way that is natural, convenient, rigorous, and implementable. Although none of the more common programming models and notations allow the representation of concurrency, there are many programming models that do allow the expression of communication and concurrency, and deserve study. The Homogeneous Machine is our vehicle and stimulus for these studies.

What appear to us to be the most promising programming models are those in which a program consists of a number of sequential processes that communicate by explicit message passing. The models of this type range from fairly simple models such as Hoare's CSP notation to extraordinarily general approaches such as Hewitt's Actor model. Our focus has been directed by the practical efforts in programming a real system, and is between these two extremes. Dick Lang's study [Lang 5014:TR:82] of object-oriented programming notations for homogeneous machines is typical of this middle road. While this approach is believed to be promising, it would involve a major effort to reduce to practice at this time.

One set of simpler approaches under early investigation are generalizations to CSP in which communication is modeled as queued, as it is in the actual machine. Such approaches can be implemented quickly and easily in conventional programming notations by embedding, thus allowing programming experiments under a model to be investigated without the laborious process of building a complete language processor.

The fundamental difficulty observed in styles that use less restrictive synchronization in communication than CSP is that while any number of example programs can be written and checked by simulation, we have every reason to believe that the complexity of large and very concurrent programs will not be effectively managed by ad hoc methods. There are a number of structured communication patterns, derived from self-timed systems closures, that model weak synchronization in queue-connected communication, but these also appear to be excessively restrictive.

Here, then, is a review of some of the programming experiments underway.

2.2.4.2 Minos

Craig Steele (Advisor: Randy E. Bryant)

Craig Steele was partially funded last year by a Silicon-Xerox fellowship.

This work has concentrated on the specification of the MINOS message-passing operating system for the hypercube processor ensemble, to be implemented in an extended Pascal dialect.

MINOS is designed to provide a minimal but general framework for the computations on the Hypercube. It is intended to support a model of computation composed of many cooperating processes, of arbitrary

physical location, dynamically created and linked by a root process. Multiprocessing on each physical processor is normal; multiple computations are feasible.

The logical and physical aspects of process communication and port interconnection are separated as much as possible in the proposed implementation, to provide both reasonably convenient and intelligible high-level language usages and efficient message transmission during actual communications. Fairly complex initialization procedures are to be effectuated by (generally remote) daemon "namer" processes. Subsequent usage of those communication channels will be calls to the local physical processor input/output kernel.

Though the initial implementation of MINOS will be oriented to explicit static assignment of processes to physical processors, the mechanisms employed anticipate the extension to a more dynamic system of computation by elaboration of existing components, in particular the namer and performance monitor.

MINOS will be implemented primarily in an extended Pascal, which will also be available as an application language. This implementation supports separate instruction and data address spaces, which is prerequisite for code sharing and consequently greater numbers of identical independent processes.

An embedded language allowing dynamic process creation and interconnection is being defined for the Pascal system.

A Pascal cross-assembler for the 8086 was acquired, and is being bootstrapped onto our VAX. The runtime system, including an interactive debugger, will be customized for smooth interface with our operating system, so that Pascal will be a well-supported user programming language as well as a system language.

An alternative node design for homogeneous machines, with a separate communications processor, has also been studied. A communications processor specialized for serial communications in a binary N-cube interconnect scales well to higher values of N due to its use of N+1 independent serial busses. Arbitration is accomplished serially as well, requiring only two pins for the entire chip. The communication processor will relieve the "working" processor of the node of any overhead required for forwarding of indirectly routed messages.

2.2.4.3 Combinator Expression Reduction System

Bill Athas (Advisor: Chuck Seitz)

Caltech's efforts in experiments with Homogeneous Machines can be described in one perspective as an attempt to more effectively match or pair off processor area to storage area. If Homogeneous Machines can be evaluated in this context, then the logical extreme is to attempt to associate a processor with each storage element. This model has the interesting feature, however implemented, that data does not leave storage to be operated upon. Certain computational arrays and content addressable storage are examples of computing based on this model. A more readily programmable implementation of such a computing structure may be approached by considering combinator expressions.

While most modern programming languages are derived from the von Neumann machine model, the LISP language is derived from Church's lambda calculus. There has been considerable interest in implementing LISP and other applicative languages on non-von Neumann machines, for example, reduction engines and data flow machines. Building upon this idea, a result by M. Shonfinkel, which was brought to the attention of the Computer Science community by David Turner [Turner 77] is that lambda expressions can be translated into an alternate form called combinator expressions. Turner describes how a LISP-like language can be translated into combinators and then interpreted by a conventional computer. Two benefits directly derived from this translation is that there is no overhead for normal evaluation and the code is self-optimizing.

The process of translating lambda calculus expressions into combinator expressions requires abstracting variables to form a variable-free notation. Evaluation is accomplished by application of a function to arguments where the combinators are used to jockey arguments into position. A crucial observation about the combinators is that they are very simple in operation, for example the three essential combinators are defined as follows:

```
S f g x --> f x (g x)
K x y   --> x
I x     --> x
```

Since the combinators are rather simple, it could be feasible in a chip implementation to associate a combinator primitive with each storage element. Adhering to the structure of LISP, the elements would be arranged as a binary tree.

All of the primitives needed to define a machine have been conceptually resolved, the most complex being the paradoxical combinator 'Y', which causes a mapping of the function domain back onto itself. This combinator is dealt with by equipping each storage element with a stack. The Y combinator causes a local recursion to as many adjoining elements as necessary. The stack is also used for the pairing or CONS operator in LISP. Through the facility of normal evaluation, the tree computing structure may stop computing, even though there is more to be done. Attempting to unload the resulting data structures from the machine, for example to print a result, will cause further execution. Bear in mind that the tree is not only the program but also the resulting data structure.

In order to be able to experiment with combinator systems, a LISP to combinator expression compiler and optimizer has been written in UCI-LISP and is running on our DEC-System 20/60. Also working is a combinator interpreter with debug facility written in Pascal. The system is completely functional except the combinator needed to handle multi-variable arguments does not work correctly. Once it is operational, the system should be complete onto itself. When completed, simulations will be done comparing it to Landin's SECD machine for LISP.

If the results are favorable, a number of other implementations will be considered, including a suitable interpreter for a homogeneous machine and a VLSI implementation of the storage element.

3. Design Disciplines

3.1 Self-Timed Design

Chuck Seitz

Both the theory and application of the self-timed design discipline [Seitz 80] continue to be stimulated by design experiments in MOS medium. Our objective of producing this year one or two exemplary and LSI scale self-timed designs has been well realized by a family of self-timed communication chips. These efforts are aimed at being able, eventually, to specify as input to a "silicon compiler" the form of message passing communication desired between subsystems, including performance requirements, and have the compiler produce a suitable layout for the interface chip, or cell within a very large chip. The designs reported below already show a substantial degree of parameterization.

Although internal speed-independence is not actually a requirement for systems with self-timed interfaces, internal speed independence has been the rule rather than the exception in these "exemplary" designs. Since achieving perfect speed-independence is difficult and expensive, usually both in area and performance, exceptions are made, and create a recurrent difficulty of verifying the correct sequential operation of these asynchronous networks. The principal tools for this verification are either a form of sequence diagram or s-net [Seitz 70] or ternary simulation with MOSSIM [Bryant 5033:TR:82], either of which is satisfactory for verifying closed loop (causal) sequencing such as "input change precedes output change," but cannot deal with open loop (timing) sequencing. Investigations are continuing in this problem, which is properly one of speed-independent rather than of self-timed design.

3.1.1 A Self-timed Chip Set for Multiprocessor Communication

Douglas Whiting (Advisor: Charles L. Seitz)

Partially supported by an NSF fellowship.

An initial aim of this project [Whiting 5000:TR:82] was to determine what type of self-timed building blocks, along the lines of Macromodules [Clark 67], might be useful for the VLSI age. One of the first conclusions reached was that since small synchronous processing systems can be built cheaply and reliably in today's technology, any immediate contributions from the self-timed discipline would have to come at a higher level than that of a small system such as a microprocessor. Where people start getting into timing and system integration difficulties is in making multi-microprocessor systems, an application directly connected with other research interests at Caltech, and that apparently accounts for a significant and growing fraction of the microprocessors produced today.

Self-timed signalling schemes have many advantages over synchronous designs in connecting multiple processing units, since the composition of self-timed components can be specified entirely by the interconnection topology, without regard to the electrical parameters

that determine worst-case timing. Thus it is much simpler to construct large computing systems if each subsystem has a speed-independent interface.

Historically, the most significant attempt to define a speed-independent processing framework was the macromodules. The internal logic of the modules was implemented in standard ECL, mostly SSI, and each macromodule was a small box which could be inserted into a rack. Typical module functions were arithmetic operations, Boolean operations, memory, registers, and control flow (forks, joins, etc.). The overall system function was determined wholly by the topology of interconnection, some of which was implicit in the placement within the rack and the rest of which was explicitly determined by data and control cables. One nice feature was the ease with which concurrent processing could be implemented. No knowledge about timing constraints was needed, and the only rule was, "if it fits, it works." Unfortunately, the concept of macromodules does not map directly to one macromodule equals one VLSI chip, because the functional level of a macromodule is quite small compared with a modern chip, and the number of interconnections required is very large.

More recently, various speed-independent bus schemes have been proposed to allow independent processors to communicate with one another. An example is the TRIMOSBUS, which allows for any sender on the bus to send a message to one or more receivers, and wait for the last of them to respond.

One product of this research is a bus communication technique which is quite flexible and (for the most part) transparent to the processors. A typical application is for message communication between microprocessors. Each sender specifies a destination for its message, and, since receivers may have shared aliases, there is the capability of one-to-many communication.

The bus is speed-independent. On each data transmission cycle of the message, the sender interface waits until all participants have signalled receipt of the data before initiating the next cycle. By employing such a speed-independent signalling scheme, any number of ports may be added to the bus without loading problems. Sendership arbitration is included as an integral part of the signalling scheme, incurring very little overhead and providing a measure of fairness. The protocol allows for one-to-many communication in which the sender must wait for all receivers to respond to each datum transmitted. The width of the data bus is arbitrary, and only three control wires are necessary for normal transmission cycles.

The system design includes both an interface chip, and also an "F" (for filter, or forward) chip that allows the bus to be divided into several local buses. The presence of an F chip is entirely transparent to the processor software. Thus the bus topology may be reconfigured to match the communication pattern and load using these chips as building blocks.

In addition to the goal of finding useful self-timed building blocks, another purpose of this research was to produce an exemplary self-timed design in MOS. The bus specifications were devised in a top-down fashion: first, the global bus structure, then the signalling

conventions, and finally the implementation in MOS. However, the MOS level design was made self-timed from the bottom up: only in a few instances were concessions made to chip area by invoking a knowledge of the actual timing involved. Much effort was put into verifying that the design was truly speed-independent, and this experience has provided insight into the type of tools needed for functional verification of such designs.

From this complete logical design, a group of students in the VLSI design laboratory course have produced a design of the interface chip, unfortunately not quite complete in connecting the independent parts of the design.

3.1.2 FIFO Buffering Transceiver

Charles H. Ng (Advisor: Charles L. Seitz)

The aim of this project [Ng xxxx:TR:82] is to develop the communication channel needed to connect any two processors on a one-to-one basis within a homogeneous machine. This chip is a direct descendant of the bus communication chip described above.

Each processor of the homogeneous machine (qv section 2 above) must be able to communicate by passing messages to other processors in the ensemble. There are several ways in which messages can be passed, and good reasons that the communication network of the homogeneous machine is implemented with a large number of one-to-one channels.

Messages can be broadcasted on a contention network, such as the ethernet, or on a bus by the sending processor to the entire system. The destined receiving processors then pick up the messages. This approach is severely limited by traffic congestion in large networks, unless a mechanism is provided for separating the communication medium, filtering out local messages, and forwarding non-local messages. Chips for this kind of filtering and forwarding communication system have recently been investigated by Whiting [Whiting 5000:TR:82], and his work is greatly influenced the work reported here.

Instead of this "one-to-many" linkage, processors can be linked on a "one-to-one" basis. Each processor has a number of channels to communicate with its neighboring processors. In order to send a message to a distant processor, the sending processor first sends the message to a processor to which it is connected (a neighbor), and the neighboring processor sends it to the next neighboring processor. This process continues until the message is received by the destination processor. This method calls for a routing algorithm for sending messages in order to achieve high efficiency and to avoid deadlocks. Moreover, there are the questions of network topology, required bit rates, packet sizes, and buffering requirements. In answering these questions, we have been guided largely by the simulations reported in [Lang 5014:TR:82].

The resulting chip, called a FIrst-in-first-out Buffering Transceiver (FIBT), provides a full duplex communication channel between any two processors. FIFO queues are provided for buffering data on each communication channel. FIBT accepts data packets from the host processor via a parallel data bus and serially sends them out to its

communication partner. FIBT handshakes with the processor by using asynchronous interrupt signals.

Linkage between any two FIBT is accomplished by using only two wires. Both data bits and handshaking signals are sent by these two lines. The FIBT system is neither synchronous nor completely asynchronous; instead, it is a system in which the clock signal is used as a frequency reference whose phase is arbitrary, much as in RS-232 data communication. The start and stop bits in the data establish the phase.

Finally, FIBT is implemented in nMOS technology. The design is parameterized so that data packets of various sizes can be handled. The layout of the chip is coded in Earl. Any member of the family of chips can be produced by changing three basic parameters.

Three layouts have been done for the FIBT. The first layout is a non-parameterized FIBT which carries two 16*32- bit FIFOs. The second layout is a parameterized FIBT with the same FIFO size. The third layout is done by changing the parameters of the second layout to generate a new FIBT with two 8*10-bit FIFOs. The first layout took about three man-months to do, and the second one took an additional man-month. In contrast, the third one only took about one man-hour. With a little extra work, a whole family of FIBTs can be generated. Finally, several versions of the FIBT are being fabricated by MOSIS. We expect to see these chips returned and tested soon.

3.1.3 Self-timed System Notations

Jo Ebergen (Advisor: Chuck Seitz)

A visiting graduate student from Eindhoven, Jo Ebergen, developed a linearized version of the s-net notation [Seitz 70] for signals that also parallels the test language developed by DeBenedictis [DeBenedictis 4777:TR:82]. This notation allows a concise statement of closure demonstrations, for example:

C1 = [a:=1, b:=1; 1=:x; a:=0, b:=0; 0=:x]

C2 = [x:=1, c:=1; 1=:d; x:=0, c:=0; 0=:d]

represents an interconnection of 2 2-input C-elements, both function and domain (environment), with inputs a,b, output x of C1, and inputs x,c, output d of C2, and

C3 = [a:=1, b:=1, c:=1; 1=:d; a:=0, b:=0, c:=0; 0=:d]

represents the function and domain of a 3-input C-element. The composition (C1 o C2) does not equal C3. This is correct, since the domain of (C1 o C2) is broader than that of C3. However, because the composition (C1 o C2) o C3* = [], where * (dual) represents exchanging system and environment, or function and domain, and [] indicates liveness, (C1 o C2) is shown to be a valid implementation of C3. A new proof of the weak conditions theorem [Seitz 80] was expressed in this notation.

It is our hope that a generalization of this notation may be useful as a message-passing programming notation as well as for signal-described systems.

3.2 CMOS/SOS Technology

Chuck Seitz

The inclusion of CMOS/SOS design as part of a renovation of the Caltech VLSI design laboratory course [Seitz 82b] is now entering its third year. In this full-year course the students individually design one small CMOS/SOS and one nMOS project in the first quarter, and in groups design one to several larger projects, thus far only nMOS, in the winter and spring quarters. Although the students learn from the beginning to do designs and layouts in both CMOS/SOS and nMOS, the initial emphasis is in CMOS/SOS design, since it is somewhat simpler than nMOS, and can be treated with a simple switching model. Because the CMOS/SOS designs are the first ones done by the students, and the fabrication schedule has been problematical, these designs have all been relatively small.

CMOS/SOS chips designed in previous years have been fabricated at Hughes and Rockwell, with fabrication arranged by the Caltech Jet Propulsion Laboratory, and at Asea Hafo (Sweden) under a no cost agreement. Some of the chips returned have been tested, and have verified the basic design style.

A trip to RCA by the MOSIS project leader, Danny Cohen, and Chuck Seitz in July 1982 resulted in an agreement for RCA to provide CMOS/SOS fabrication services through MOSIS. A new set of geometrical design rules and electrical characterization of this CMOS/SOS process was established in the discussions with RCA, and Chuck Seitz has prepared an initial CMOS/SOS technology file for MOSIS with this information. The first fabrication run is scheduled for mid-November, and the second in February. If these runs go well, we expect to attempt some much larger CMOS/SOS designs this spring.

3.3 Testing

Erik DeBenedictis (Advisor: Charles L. Seitz)

The approach to testing that has been investigated has attempted and largely succeeded in merging hierarchical structured design techniques with testing [DeBenedictis 4777:TR:82], [DeBenedicitis 82]. This approach tests a system by testing each of its parts and verifying the connections between the parts.

A highly structured test language with abstractive capabilities corresponding to common abstractive design practices has been defined. The test language is not a general purpose language. Test language

procedures can be analyzed at the source code level and converted into a form amenable to efficient execution on a high speed tester. Some of the information extracted from the test language procedure is the length of the test sequence and the exact effect of the arguments upon the resultant output sequence. The highly structured nature of the test language allows it to be interactive, as the FIFI test system demonstrates.

The abstractive primitives of the test language correspond to the process of hierarchical composition in design. The basic scenario of the test language is as follows: 'Given a number of parts, and tests for each of the parts, determine how to test the parts when they are combined into a system.' The abstractive primitive is called an 'access procedure' and describes how to translate a test for a part of a system to a test that can be applied to the entire system.

A tester was constructed as a research effort into new tester design styles. The tester was of a very small sort, based upon a microprocessor. The data paths of a high performance tester were implemented as a program for the microprocessor. This approach allowed experimenting with different forms of data paths and instruction sets without hardware changes. The cost of this flexibility is that the tester is very low in performance.

In addition to being a research vehicle for tester design, the small tester was a demonstration of a tester that could be distributed to individual engineers (or students) at low cost. The small tester is software compatible with a high performance tester (if one were to be built) and could be used to interactively develop tests to be run on the high performance tester. The development of test patterns is not an activity that requires a high performance tester, but does require a great deal of dedicated time with a tester. Using a $5,000 tester instead of a $5,000,000 tester is desirable whenever possible.

A test language interpreter was written in SIMULA to run on a DEC-20. The interpreter implements the complete test language, and is interactive. The interpreter can provide several types of output: (1) binary codes to drive the small tester, (2) an output form similar to the test matrix input of commercial testers, and (3) a timing diagram form suitable for documentation.

The characteristics of a tester that could execute the test language efficiently have been explored. The design of such a tester is very similar to commercially available testers. Some design differences are apparent, however:

- Commercial testers are generally irregular. For example, a tester may have 1024 vectors of high speed storage and no provision for expanding the storage in a homogeneous manner. The tester design proposed can generate arbitrarily long sequences of test vectors in a homogeneous manner.

- Commercial testers are generally designed for a fixed timing scheme. For example, a tester may provide a hardwired 1.5 phase clock and be capable of running at only a fixed rate. The tester design proposed allows clock signals to be generated like any other

signal, hence the number of clock phases is limited only by the number of pins, and the timing for each test step should be individually specifiable.

A design-for-testability strategy is implicit from our approach to testing. The method is somewhat more abstract than conventional design-for-testability strategies; it is a method that can be used to describe many existing design-for-testability strategies. (All scan path and combinations of scan path and other techniques can be described, but self-test, syndrome test, and test methods relying upon obscure electrical properties cannot.)

The design-for-testability strategy classifies parts into two types: composition systems and primitive parts. Composition systems can be broken down into a collection of parts. Primitive parts cannot be broken down. Tests for primitive parts must be generated by other means. Typical primitive parts are combinational logic, PLAs, RAMs, and ROMs. Test generation for these types of parts are known. The necessary test information for each composition system consists of an access procedure that can access each of the parts internal to it.

The test language allows tests to be generated in the same structured manner that designs are generated today. Several persons can generate test information each for a particular part of a design. The communications between these persons is minimal, and is formalized by the test language.

A catalog of testable parts and compositions could be created. Each entry in the catalog would consist of (1) a logic-diagram specification of the part or composition, (2) an access procedure for that part, and (3) timing specifications for the inputs and outputs. A designer would have testability of a system guaranteed if he used only testable parts and the timing specifications of the parts matched over interfaces.

The FIFI test system demonstrates an interactive test system for design characterization and the development of tests.

4. Design Tools, Workstations

4.1 Design Tools

4.1.1 Switch-Level Simulation

Randy Bryant

Partially supported by the Silicon Structures Project.

We have recently completed the implementation of MOSSIM II, [Bryant 5033:TR:82], switch-level simulator with greater generality and capabilities than previous programs. This program is written in Mainsail(TM) and is available for distribution, along with a set of related programs for hierarchical network specification, circuit extraction, and network comparison.

A switch-level simulator models an MOS circuit as a network of transistor "switches" with node states 0, 1, and X (for unitialized or invalid) and transistor state "open", "closed", and "indeterminate". It can accurately model a variety of circuit structures such as (bidirectional) pass transistors, static and precharged logic, busses, and both static and dynamic memory. MOSSIM II has a very general network model in which every transistor can be assigned a "strength" to indicate its approximate conductance relative to other transistors which may form part of a ratioed path, and every node can be assigned a "size" to indicate its capacitance relative to other nodes with which it may share charge. Almost all of the circuit structures seen in MOS digital circuits today can be expressed in this network model. Furthermore, parts of the system can be specified in terms of their functional behavior in the form of a Mainsail module.

Networks for simulation can be derived from several sources. First, the network description language, NDL, embedded in Mainsail, allows the user to describe a network as a hierarchy of subnetworks with provisions for generating unique names for every node and for collecting sets of nodes into vectors. An NDL program is expanded into a network of transistors prior to simulation. Second, a circuit extraction program based on Clark Baker's program at MIT can derive the network directly from the CIF mask description. We have enhanced Baker's program to handle general wires and polygons for extracting curvilinear (Boston) geometry. Circuit extraction provides a very reliable, if somewhat brute force means of checking a design as it will actually be fabricated. However, it is often difficult to debug a design this way, because most of the node names are lost in the process, and the user has limited ability to correct the errors in the design.

As an alternative to simulating a network derived by extraction, the user can first design and debug a circuit using a more top-down description such as in NDL, and then check the extracted circuit against this description using the network comparison program MOSDIF. Our program attempts to determine whether two transistor networks are isomorphic using the O(n log(n)) automorphism partitioning algorithm described by Kubo, Shirakawa, and Ozaki (ISCAS, 1979). This program can return with one of three answers: 1. the networks are definitely isomorphic, 2. the networks are definitely not isomorphic, or 3. no

mismatches were found, but an isomorphism could not be established. This latter case generally happens only when the networks themselves have automorphisms (e.g. the memory cells in a random access memory array), which can be resolved by having the user specify a matching of key nodes in the two circuits (such as the inputs.) Furthermore, the user can forcibly match nodes which are not equivalent to help pinpoint the differences in the two networks. This program works very fast (about 10 seconds for a 2000 transistor network), and hence this process of specifying matches and recomputing the partitioning can be performed interactively.

The simulation capabilities of MOSSIM II include the ability to define a clocking scheme; to look at or set the state of any node in the network; to set breakpoints based on the network state; and to drive the simulator by user-written Mainsail procedures. MOSSIM II can also apply more stringent tests of a circuit beyond its normal unit delay, switch-level model. It can use ternary simulation to detect potential timing errors. Unlike conventional timing simulators, ternary simulation tests whether the design will operate correctly for all possible circuit delay parameters. Ternary simulation can be augmented to check for potential errors caused by "dynamic charge sharing", i.e. glitches caused by transient charge sharing effects. MOSSIM II can also be run with charge storage disabled (for static circuits), with limits placed on the charge retention time, and with checks for unrestored logic levels (for CMOS). All of these tests use the X state to represent various forms of bad behavior and rely on the relatively fast and accurate X propagation algorithm of MOSSIM II to keep track of the effect of a potential problem. Tests of this form help establish that a design will function properly for a wide range of processing parameters and operating condition -- a task which is difficult to do by more detailed circuit simulation.

4.1.2 Earl -- An Integrated Circuit Design Language

Chris Kingsley (Advisers: Carver A Mead, Chuck Seitz)

Earl [Kingsley 5021:TR:82] is an integrated circuit design system with several novel features, but its central characteristic is that it supports a separated hierarchy design method. Earl has just two kinds of cells: leaf cells contain only geometry, and composition cells contain only other cells.

Earl has evolved from an attempt in 1980 to include port definitions and constraint solving in an extension of LAP. Because the Simula base for this tool was not sufficiently robust and had a very restricted name and address space, the basic ideas were incorporated into a design language which was implemented in C starting in September 1980. Earl has continued to evolve from its first use in chip designs in the Spring of 1981, and with extensive student use to its present stable form by June 1982. Earl supports both nMOS and CMOS/SOS designs.

Numbers in Earl are complex, with the real and imaginary parts representing x and y coordinates. Much of the flexibility of Earl as a programming language results from its extensive use of lists, for example, to group together related points. Earl supports all of the usual mathematical and logical functions, and control structures.

Recent additions include file input and output, which is helpful for programs such as PLA generators.

The Earl specification of a leaf cell includes:

- Cell declaration. Cells may have parameters, including list parameters. Example: "cell placore(there);"

- Port declarations, and declaration of any internal points. Example: "west group vdd, in2, gnd;"

- Constraints on the ports and internal points. Example: "ycon south |=0| gnd |>7| in2 |>3| vdd |=0| north;" Notice that since Earl treats x and y constraints separately, it is not really a compacter. However, by extensive and tricky use of constraints one can code leaf cells to accommodate variations in geometrical design rules.

- Geometry specification. Example: "island wire in2, @+2; miss (incont, -5); seg pcon;"

The principal composition operators are called "horiz" and "vert", and allow one to compose previously defined cells either horizontally or vertically, with ports connected (if possible).

When Earl is asked to make the cif or to plot a cell, it first builds a constraint graph based on the composition operators, with values associated with the arcs from the constraint section of the leaf cell definitions. The constraint solution, if there is one, determines the geometrical coordinates of all of the declared ports and points. The constraint solution will be to a locally minimal set of distances, and circular constraints are harmless so long as there is a solution. Next, the code associated with the geometry section of the cell definition is executed for each instance type to generate the cif symbols. Then the cif calling these symbols is built.

The Earl we use at Caltech normally has contacts defined at circular symbols. This feature, together with the "miss" function noted by example above, provides a basic facility for producing "Boston geometry." There is otherwise nothing very Bostonian in Earl's character; he is equally comfortable living in Manhattan or other cities.

Earl has now been distributed to about 40 universities and companies, and we have received many favorable comments about his successful designs and robust character.

4.1.3 A Hierarchical General Interconnect Tool

John Y Ngai (Advisor: Chuck Seitz)

The complexity of VLSI circuits dictates a hierarchical approach to its design and layout. The chip as a whole can be viewed as composed of several simpler modules which in turn are composed of simpler modules. The compositions recur until the modules arrived at are simple enough to be understood, designed, and layed out comfortably. Such an approach relies heavily on the ability to compose modules successively and efficiently.

The geometrical or layout aspect of composition requires one to connect the appropriate signal ports of the modules together to form a super-module which may or may not has its own signal ports. Design tools such as Earl [Kingsley 5021:TR:82] support only connections by abutment, which requires exact match of signal port sequence among the modules. Stretching of modules to enforce coordinate matching of ports is accomplished by constraint solving. Although this approach works well at the lower and more regular levels of design, it is very poorly suited to the last several assembly stages of a large chip, in which one is trying to route a large number of signal wires, power, and clocks, between signal ports that are geometrically very much constrained.

It is obviously very much easier at these higher levels of assembly to connect signal ports by routing signals to the required port positions. The goal of this project, then, is to produce a hierarchical general module router. In addition, this routing tool is intended to explore the general strategies for using multiple layer interconnections.

The work is somewhat different in emphasis than other work we are aware of in routing, in particular:

- Routing is performed in raster order, based principally on local information. This approach is an extension of a raster-order routing technique developed and implemented for printed circuit boards by Oestreicher and Seitz in 1971, and which performed assignment, placement, and routing together with surprisingly good results and high efficiency.

- Based on a model and analysis by Mead and Rem, [Mead, Rem 81], of the use of additional thicker metal layers in VLSI circuits, one can expect these upper layers to be used preferentially for progressively more and more distant communication on a chip. A designer may approach this optimal use of additional layers naturally by restricting the lowest levels of design to the lowest mask layers. For example, a set of cells restricted to diffusion, poly, and first metal could be wired together by routing using first and second metal. Notice that second metal can run over the cells, and make contact to internal ports (first metal areas) inside the cells, while first metal would be constrained to the wiring areas between cells and could connect only to ports at the periphery of cells. This same strategy can then be used to connect such modules using only 2nd and 3rd layer metal.

What has been accomplished so far is:

- definition of a standard input specifications for the router, and its interfaces to other design tools.

- design of the various algorithms that achieve the routing requirements.

and the work in process is the implementation of a system that can be put to experimental use this spring.

4.2 Workstations

Dan Whelan, (Advisors: Carver A. Mead, Jim Kajiya)

Partially supported by the Silicon Structures Project

Our work with CAD graphics systems has been based on the premise that bit planes can be most effectively used to represent separate logical layers such as those that might be used in a sticks system or even the logical mask layers that are used to represent physical process layers. If this is done properly, bit planes will represent disjoint logical layers such that the update of these logical layers can be represented as separate processes. This suggests that some parallelism can be exploited by the graphics system in the update of these bit planes. Therefore, we set about designing a low cost graphics system that provides graphics processing capabilities on a per bit plane basis.

We currently have a prototype system running, [Whelan 4334:TR:81]. The design is based on a graphics processor chip manufactured by the Nippon Electric Company. The system is configured out of two types of boards. The bit plane board implements four identical bit planes, each with 128K bytes of RAM and a graphics processor which can be used to fill simple areas and draw lines. With 128K of RAM, each bit plane can be treated as a 1024 by 1024 bit map of which some 640 by 480 is visible. Hardware panning can be used to view the rest of the display. Providing such large amounts of off screen memory may prove extremely useful, since the programmer can view the bit map as one large image or several smaller disjoint images. The other type of board provides a 4096 by 24 bit color map and also generates the system clocks. Both boards are Multibus compatible and are being used with the SUN processor and Ethernet boards.

Interestingly, even though the architecture of the NEC graphics display controller chip severely limits its performance, the ability to use up to 28 of them in parallel enables the system to perform better than a large number of conventional systems. This architecture with the right kind of display processor shows a lot of promise for the future.

5. Concurrent Architectures and Algorithms

5.1 Signal Processing and Scientific Computing

5.1.1 Concurrency in Linear Algebra Computations

Lennart Johnsson

5.1.1.1 Algorithms

Matrix and vector operations such as the computation of inner products, matrix-vector products, matrix-matrix products, and the solution of linear systems of equations are an important part in many signal processing applications. These operations are classical in the area of numerical linear algebra. A large number of algorithms have been devised and analyzed, in particular for sequential machines. Concurrent algorithms have also been devised and used for machines with a few processing elements.

Algorithms, even of the concurrent type, have largely been devised with the model that communication is for free. Data can be made available to any processor without delay whenever needed. This model of a system is, in our opinion, unrealistic. Different models of the relative cost of communication to computation applies at different levels of system modeling. What is a proper model at a given level is determined by factors such as design discipline, architecture, and communication protocols. Models used for different levels of system representation have to be consistent with each other and with the technology used to implement the system.

Our work on concurrent algorithms for linear algebra computations is focused on the topology of algorithms, communication requirements, node complexity, and the mapping of concurrent algorithms onto various computational structures. The efforts during the last year have focused on algorithms suitable for multiprocessor machines with a very limited complexity and fanout. The cost of a communication is assumed to equal that of a local memory reference. The node can be characterized as having (in most cases) only a few registers for data and an arithmetic unit. We call multiprocessor systems of this sort computational arrays. They fit within the intuitive notion of systolic systems, [Kung, Leiserson 80]. In some of our earlier work on sparse matrices a considerably more powerful processor and operating system is required. In sparse matrix operations as well as in many problems of scientific computing the management of dynamic data structures is crucial for efficiency.

The term computational array is used to convey the notion that the programmability is indeed very limited. The functionality is bound at design time to a few, high level operations. The set of possible operations for a computational array may well include matrix-vector and matrix-matrix multiplication, and a few different methods for solving linear systems of equations within the realm of the processor complexity stated above. Instructions and/or data are pipelined through the array. Even though broadcasting is undesirable in VLSI there is a trade-off in the implementation of some of these algorithms. In algorithms having a "turbulent" data flow, i.e., data flows in loops, pipelining may cause

processors to be idle for some cycles awaiting partial results to traverse a loop, [Johnsson 4287:TR:81].

During the past year algorithms have been devised for various array configurations and computational problems. Part of the effort has been devoted to the mapping of fully concurrent algorithms, i.e., algorithms using the maximum number of processing elements for a given topology, onto arrays having fewer nodes than the graph of the fully concurrent algorithm. The mapping study has been restricted the case where the array is isomorphic to a subgraph of the graph of the algorithm.

The particular methods of computational linear algebra for which concurrent algorithms have been devised are: LU-decomposition based on Doolittle's, Crout's, and Cholesky's methods, [Johnsson 82a], QR-factorization based on Householder transformations, [Johnsson 82b], and Given's rotations, [Johnsson 82c], the conjugate gradient method for solving linear systems of equations, [Johnsson, 5040:TR:82], the Discrete Fourier Transform as well as the Fast Fourier Transform for its computation, [Johnsson, Cohen 82a], [Johnsson, Cohen 82b].

Most algorithms devised have the characteristic that the time complexity of the concurrent algorithm equals 1/M times the complexity of its sequential counterpart, where M is the number of processors in the system. It is assumed that interprocessor communication is equivalent to a reference to local storage. Householder transformations and the conjugate gradient method like Gaussian elimination with partial or total pivoting requires global communication in each major step of the algorithm. Concurrency is limited by sequential requirements of the algorithm, global communication and the restriction of the machine to have nearest neighbor connection in a plane. The machine topology studied has typically been a regular graph of degree 3, 4, or 6, but Boolean N-cubes and trees with the leaves interconnected to a ring have also been subject to investigation.

If a fully concurrent algorithm has to be instantiated partially in time, partially in space, in order to fit on a given array, the task is how to distribute the storage and computations among the processors of the array. The total amount of data storage is essentially determined by the storage required to describe the problem in the examples we studied. The data storage requirement can be made independent of the problem partitioning with suitable algorithms. The storage can either be external to the array of computing elements, or distributed among the processing elements, or a combination thereof. Block algorithms as well as sweeping algorithms can be devised in either case. An example of a block algorithm is given in [Johnsson, 4287:TR:81]. Sweeping algorithms are described in the same report and [Johnsson, 5040:TR:82].

5.1.1.2 Formal treatment of computational arrays

The formal description of concurrent algorithms, and the verification of their correctness is of considerable importance. Our attempt to a formal treatment of concurrent algorithms aimes at algorithms that are typically described in a mathematical notation, and that are feasible for a direct implementation in hardware. This effort is carried out in cooperation with Danny Cohen at USC/ISI. A notation that explicitly models storage is used. The notation is of the traditional mathematical variety, but expressions in the notation can be given an interpretation in the implementation domain. Algebraic transformations of expressions in the notation can be carried out in the usual manner in order to derive implementations of different characteristics that are guaranteed to be correct.

Control signals are modeled in the same manner as data flow. Operations on control signals are described in the same manner as operations on data.

The notation has so far been tried on a few examples, such as matrix-vector multiplication, matrix-matrix multiplication, FIR filters, the Discrete Fourier Transform, and implementations of the Fast Fourier Transform, [Johnsson et. al. 81], [Johnsson, Cohen 82a], [Johnsson, Cohen 82b].

Used for verification, expressions directly describing an implementation can by algebraic manipulations, be transformed into a form that directly describes the input-output transformation of the entire array, even when explicit control is used to sequentialize the computations.

5.1.2 Computer Arithmetic

Chao-Lin Chiang (Advisor: Lennart Johnsson)

The binary number system has an inherent drawback, the carry chain. In the residue number system addition, subtraction and multiplication can be performed on each individual digit independently. There is no carry. Possible applications can be found in signal processing and algebraic computations.

We have undertaken a study of the possible merits of the residue number system if implemented in VLSI. Only preliminary results are available at this time.

Several different ways of implementing a multiplier in the residue number system has been investigated. The following three alternatives are all of interest

- a table look-up method
- a combination of tables and adders
- arrays of full adders

After an analysis of chip area and delay time for each alternative we have concluded that for a base p that can be represented by 4 bits or less a table look-up method is preferable, that the second alternative

offers advantages in the range 5 - 9 bits, and the third alternative is competitive when more than 9 bits are required.

For a given dynamic range there are, in general, several possible choices of moduli. Once our estimates of area and performance for a given moduli become firm, it should be possible to specify what set of modulus is optimal.

The following basic properties of the residue number system should be noticed:

- A residue number system is more efficient than a binary number system in the sense that if a given number X requires m bits in a binary representation, then the same number of bits can represent a number up to ~1.44X in a residue number system.
- The code efficiency approaches 1 as $1/\log_2 X$.
- There exist designs of N bit residue multipliers with O(NlogN) area and delay time in the range $O(\log_2 N \log_2\log_2 N)$ to $O[(\log_2 N)^2]$ depending on which timing model is used.

5.1.3 Synthetic Aperture Radar

Sheue-Ling Lien (Advisor: Jim Kajiya)

Partially supported by the Silicon Structures Project.

The design of a Synthetic Aperture Radar (SAR) chip is used as a test case in the development of a Silicon Compiler for signal processing applications.

SAR is a radar system that through processing of the return radar signal achieves the effect of having a larger (virtual) radar aperture than the one provided by the physical dimensions of its antenna. The amplitude and phase of the received signals are collected for a certain period, after which the signal is processed. High range resolution is achieved by the use of wide bandwidth transmitted pulses. High azimuth resolution is achieved by focusing, with a signal processing technique, an extremely long antenna that is synthesized from the coherent phase history. The processing consists of a weighted summation of equally spaced samples from the signal history.

Our SAR design is based on the requirements set forth by the Jet Propulsion Laboratory of Caltech for a real-time application. Pulses are generated between every pulse period Tp (Tp=1 mS). Between each echo 1024 samples are recorded with the sampling period of Ts=50 nsec, at a delay time period of Tw. The return signal is digitized according to the range bin j and the echo (pulse) i it is in, by 10-bit words with 5-bit real part and 5-bit imaginary part.

Every 16 echoes are summed for each separate range-bin, to give an averaged result of the original signal. The pre-sum computation reduces the number of points needed in the convolution. The averaged signal, having 1024 range channels, is subject to a 40-point convolution computation. Adding signals coherently synthesizes the aperture.

Block diagrams have been made up for SAR systems based on, either static data, dynamic kernels or static kernels and dynamic data. The basic function blocks needed are FIFO shift registers, rotators, adders, and a huge amount of storage. The implementation of all the function blocks are contained in a cell library. Once the floor plan is defined the layout can be generated by using a general router to wire the function blocks together.

A dynamic shift register cell deigned by Tony Bell, Fairchild representative in the Silicon Structures Project, is used for storage. The cell needs to be refreshed at least every 0.1 micro sec. A parallel architecture is used to meet the throughput requirement. 18-bit wide registers are used for the accumulation in the convolution computation. The burst rate of data coming in is 20 MHz. 1 MHz is the rate of the convolution computation. A pipelined Cordic rotator is used in the convolution computation. The convolution requires 20 processors, each with 1 rotator, 1 adder, and 18x1k storage. Two adders are needed for averaging the input data.

5.2 Image Processing

5.2.1 Fingerprint Recognition

Barry Megdal (Advisor: Carver A. Mead)

Early work supported by ONR. Later work supported under this contract and by a Hughes fellowship.

The work done in the past year has been in the area of architectures for VLSI implementations of high-bandwidth algorithms for processing digital images, with particular emphasis on the encoding and subsequent comparison of fingerprint images.

Fingerprints are an attractive choice as the sample images for use in this research, as they are structured yet non-trivial, and fingerprint recognition is an area in which much effort has been expended, particularly in terms of recognition of relevant print features by humans. Fingerprints consist basically of a series of segments (ridges), some curved, and some straight, with ridge ends and ridge forks more or less randomly distributed throughout the images.

A successful automated fingerprint recognition system must have several properties. Most importantly, it should be capable of correctly identifying a given print to a high degree of accuracy, and be very unlikely to decide that it recognizes a given print, when in fact that print is from a different finger. It is also necessary that it only infrequently reject a valid print. These desirable properties must be maintained even in the face of the inevitable noise and distortion introduced in the process of inputing the images.

The fingerprint comparison process take place in two phases. In the first phase, parallel image processing algorithms are used to produce a graph structure encoding of the patterns of the low level "features" (ridge forks and ends) in the print image. In the second phase, the labeled graphs that result from phase one are "compared" in order to produce some measure of the similarity of the two input fingerprint

images, and therefore allow a decision on whether they are indeed from the same finger.

The image processing operations in phase one are implemented via a pipeline of specially designed processing structures, known as "neighborhood processors". Each of these processors is capable of performing operations on a "window" into the image surrounding a given pixel. Each cell in the window can communicate data to its neighbors, and is able to perform simple computations. Assuming that the data is input into the chip in raster-scan, serial fashion, the effect is to move the processing window over the entire image, pixel by pixel, in real time. The fingerprint image must undergo several such processing steps before the point at which the graph representation can be created is reached. Among these are filtering, ridge restoration, thresholding, thinning, and feature extraction.

The result of the image processing operations is the location of each of the relevant features in the print image, as well as a graph structure representing the distance relationships between the features. The process of comparing the graphs is quite interesting, as the subject has been little treated in the graphs theory literature. Algorithms to determine isomorphism of graphs are well known, but quantitative comparisons of similar but non-identical graphs is a more complex problem. The method used in this work is the computation of a similarity function, based upon the properties of the maximal common subgraphs of the two graphs in question. Much of the recent work has centered on efficient algorithms for determination of the maximal common subgraphs of two fingerprint encoding graphs, as well as the important issue of analysis of the subgraphs found for significant sets of connected nodes.

This work may be contrasted with the limited previous work in the field of automated fingerprint recognition and comparison in several ways. On one level it may be observed that the algorithms as described are inherently suitable for implementation as VLSI computational structures, while the vast majority of previous work in image processing and especially the processing of fingerprint images, has been oriented toward conventional computers. More importantly, the encoding and comparison method described are quite robust, in that they have great immunity to various forms of defects in the fingerprint images.

Two of the more notable attempts at a computer-based recognition system are the work by Wegstein at the National Bureau of Standard, and by the Hughes Aircraft Corp. Research Labs. Wegstein's system, though it used a reasonable encoding scheme, did not sufficiently emphasize the needed front-end image processing steps. The Hughes system, which was based on optical correlation techniques, proved to be too sensitive to differences in the techniques used to input the print images, as for example variations in the amount of finger pressure used.

The system as described has been implemented through software simulations of both the parallel image processing algorithms, and the graph comparison method. The results to date are quite good, with the ability to distinguish between fingerprints from the same and different fingers having been demonstrated. Some work does remain to be done in the area of establishing quantitatively the abilities and limitations of

the algorithms used.

5.2.2 A Mechanism Describing the Construction of Cortical Receptive Fields from the Center/Surround LGN Receptive Fields

Sheue-Ling Lien (Advisor: Jim Kajiya)

A study of biological vision systems was initiated this summer based on the belief that efficient, massively concurrent, algorithms for image processing systems with considerably increased capability best can be found by studying such systems. The study is carried out in cooperation with Derek Fender, Biosystems Information, Caltech.

Simple cells in the visual cortex have spatially localized receptive fields. The cells respond strongly to specifically oriented lines or edges positioned in their receptive fields. If the responses of the cells are analyzed in the spatial frequency domain, we found that the cells are tuned to specific spatial frequencies with the bandwidth of the order of one octave.

Several models have been proposed to study the receptive fields properties and response specificities of the simple cortex cells. The study by Rose proposed that the LGN cells which drive simple and complex cells in the cortex have scattered receptive fields (perhaps aligned in a row) whereas those driving cells with "hypercomplex" properties have closely superimposed fields. Daugam pointed out some defects in Rose's model that the orientation tuning curves based on this model are smooth at the top but undulate at the sides, which is contradicted by the observation of Ikeda and Wright et at.

In this research we develop a new model which describes the construction of the cortical receptive fields from a linear summation of the LGN receptive fields over the whole visual plane, based on the known facts that simple cortical cells are driven by a number of LGN cells with linear superposition properties. A two-dimensional weighting function is proposed in this model to describe the scattering of the amount that LGN inputs contribute to the cortical cells. The weighting function effectively is a linear filter over the LGN inputs that feeds into simple cortical cells.

This new model is able to affirmatively describe the spatial localization, the spatial frequency localization, the spatial tuning and the orientation tuning characteristics of the cortical receptive fields, through the variation of several parameters of the model. A graph representing the bandwidths and the orientation selectivities of several tuned cells in monkey, recorded by Devalois, is used to illustrate the control operation of the model on these characteristics. The two-dimensional spectral analysis of the cortex receptive fields, the LGN receptive fields and the proposed weighting function are also discussed in detail.

5.3 Computer Graphics

5.3.1 Ray Tracing Algorithms

Jim Kajiya

We have been exploring new graphics algorithms in order to have a supply of computation intensive applications for VLSI. In particular, we have been exploring algorithms which push the frontiers of synthetic image realism at the cost of great numerical effort.

The first result is a new algorithm for Ray Tracing Parametric Patches [Kajiya 82a]. This algorithm is not an image space algorithm in that it does not recursively divide the patches until a subpixel criterion is met. Rather it uses results from algebraic geometry to find a numerical algorithm which solves for intersections directly. Because this numerical algorithm is amenable to heavy pipelining and because the entire algorithm is amenable to massive parallelism, it is a natural candidate for a challenging VLSI implementation: We estimate that a real-time implementation requires at least 200 gigaFlops of computing bandwidth.

The second result is a new algorithm for Ray Tracing fractal surfaces. These are stochastically defined surfaces which promise to significantly enhance the realism of all computer imagery, and especially flight and tactical simulation applications -- if a real time implementation may be constructed. Our new algorithm [Kajiya 82b] can both be used for ray tracing and for scan line rendering. Current methods first instantiate the entire hierarchy of polygons and then render the scene with conventional polygon rendering schemes. This is very expensive: the number of polygons may easily reach well into the six figure range. The new method combines the instantiation and rendering phases. It proceeds by pruning the instantiation tree when a part of the hierarchy cannot contribute to the portion of a display of current interest.

The third result is of more mathematical than practical interest in graphics: Splining in non-flat manifolds. There are several applications in computer animation and in robotics in which it is desirable to interpolate positions of articulated objects. The configuration spaces of these objects forms a manifold which is most often curved. Additionally, the transformations (such as rotation) used to manipulate configurations are noncommutative under concatenation. For these reasons ordinary spline interpolation techniques do not apply. Practically, researchers simply spline on a convenient parameter space. Although the interpolation paths are highly coordinate dependent, in most cases these results are often satisfactory. The investigation here asks: "What is the proper theory?" We have found that for product manifolds the interpolation problem splits into its components, explaining why conventional spline theory is so straightforward. We develop criteria and a set of differential equations which solve the spline problem in general.

5.3.2 High Performance Graphics Machines

Mike Ullner (Advisor: Jim Kajiya)

Ray tracing works by simulating the behavior of light rays as they interact with the surfaces of a scene. It can produce pictures of objects casting shadows, it can show objects reflected in other surfaces, and it can model the refraction of light as it passes through transparent objects. Moreover, all of these effects can be achieved by using variations of the same basic computation. When programmed on a conventional computer, ray tracing algorithms are generally considered to be intractable, but the same characteristics that make them tedious on sequential machines also make them ideally suitable for concurrent operation. In particular, successive rays to be traced are almost entirely independent of each other. In a sequential implementation, this fact makes it difficult to apply the result of one ray tracing operation to reduce the computations required for the next. In a parallel implementation, on the other hand, it is possible to trace many rays at the same time because they are largely independent.

A couple of different machine organizations for exploiting the concurrency that is possible in a ray tracing algorithm have been studied. The first is a rather conventional pipeline constructed from standard, commercially available components, whereas the second uses an array of loosely coupled, custom designed processors. The array has the advantage that its performance may be boosted by increasing the number of processors, but unfortunately integrated circuit technology has not yet reached the point where it is economical to fabricate these processors. The pipeline, on the other hand, could be built with currently available devices, although its performance would be tied to the speed of its components. Either machine, however, can make ray tracing algorithms viable alternatives to some of the more conventional approaches.

Although the machines outlined above can raise the speed of ray tracing computations to a practical level, they fall short of real-time performance. They are thus unsuitable for those applications of computer graphics where real-time operation is critical and image quality is of secondary importance. Therefore, we have also studied machines for implementing other types of highly parallel algorithms to achieve real-time performance. One of these machines is designed to produce shaded images of a scene by using scan line techniques. Its performance is based more on the number of processor than on the speed of individual processors, and it may be extended to handle an arbitrary number of polygons in real-time. Another machine eliminates hidden lines from a wire frame picture in real-time. Once again, this machine is based on a collection of identical processors operating in parallel. Each of these processors consists of little more than three bit-serial multipliers together with some registers, adders, and control logic. Such a processor is feasible even with the current technology for fabricating integrated circuits.

5.3.3 Display Systems

Dan Whelan (Advisors: Carver A. Mead, Jim Kajiya)

Quite a few multiprocessor systems have been proposed for computer graphics, few have actually been built. One interesting approach has been called "a processor per pixel" and Fuchs has designed and implemented a chip for use in such a system. Our work has focused on the design and implementation of much simpler "processors" in an effort to make the processor smaller and the system practical to build.

Our display system architecture has rectangular area filling as a primitive operation, [Whelan 82]. With such a display, lines can be drawn significantly faster than with conventional pixel based display systems. More significantly, filled areas are rendered in constant time, an $O(n^2)$ speed improvement and convex polygons can be filled with an $O(n)$ speed improvement.

The processors that implement this architecture are simple extensions to ordinary RAM cells and also employ a different address decoding scheme. A 4K bit (64 by 64) device has been designed and implemented in nMOS. Currently, a microprogrammable tester is being designed to test the device. If all goes well, the plan is to build a demonstration system. Such a system ought to be able to render 66,000 rectangles per frame time (1/30 sec.). The system would be unparalleled at rendering VLSI layouts.

Currently under study are architectures that can handle on the order of 100,000 polygons/frame time. Unlike previous attempts at high performance systems, our intent is to provide for realistic shading algorithms and improved surface detailing. Such image complexity and quality is state-of-the-art currently and requires a great deal of CPU time. State-of-the-art in real-time animation is around 4,000 polygons/frame time and is achieved by some of the commercial flight simulators.

6. Formal treatment of concurrent systems, programming languages

6.1 Notations for and Analysis of Concurrent Systems

6.1.1 Concurrency in Computation

Alain Martin

This research is partially supported by the Silicon Structures Project

This research comprises three main aspects, namely:

- The study of programming notation and methods for distributed programming,
- the design and analysis of distributed algorithms,
- the design and analysis of parallel automata for these computations, in particular, large ensembles of communicating processing elements.

It is our belief that one should not investigate one of these three topics without paying careful attention to the other two. It is not possible to study programming notations and methods--in particular for expressing communication and parallelism--without trying them on significant examples. It is not possible to design and analyze good algorithms for these examples without an adequate model of a processing automaton. Similarly, it is not possible to construct highly parallel machines without a good idea of the kind of algorithms and of programming style considered for it.

During the past year, the adequacy of a distributed programming language strongly inspired by C.A.R. Hoare's CSP has been investigated. Several important changes have been made, and a semantic definition of the communication primitives has been proposed. One modification is the introduction of channel variables. (We believe that in order to apply programming formalism to a large class of computing systems, not only stored-program computers, but also, e.g., VLSI circuits, one will have to supplement the traditional notion of variable with another one describing the way in which information is stored on a conducting wire.) Another modification is the suppression of "input commands in guards," the semantics of which has been found too ambiguous and difficult to use. The construct introduced to achieve the same effect is surprisingly simple and clean: it makes the behavior of input and output commands entirely symmetrical, makes it possible to construct "fair" solutions to synchronization problems (e.g., fair arbitration) which were impossible to construct in CSP, and permits to design better solutions for certain problems than is possible in CSP.

The semantics of the communication primitives used is captured by two axioms. One, the synchronization axiom, which formalizes the synchronization properties of the primitives, is based on a previous general study of synchronization published in [Martin 81a]. The other one, the communication axiom, which formalizes the "distributed assignment" property of the primitives, is a generalization of the

substitution rule for the assignment statement.

This work will be reported in a forthcoming article [Martin 82a].

Another part of last year's research consisted in experimenting with the above programming language and semantics on a class of significant algorithms, in particular the so-called "distributed mutual exclusion" algorithms. Two classes of algorithms have been designed: for the case when the processes are arranged in a ring, and for the case when they are arranged in an arbitrary graph. (The systems are fully distributed: no central process, central clock or common store is allowed. The processes communicate with each other by exchanging messages only, and in general one process may directly communicate with only a small subset of neighbors.) We believe the solutions proposed to be novel and efficient. They suggest some new and general design and proof techniques which will require further investigation. The different solutions to this problem are described in [Martin 82b].

Not much has been done during the past year about the third part of this research, namely the design of highly concurrent computer systems. However, some applications to the general method described in [Martin 81a] and [Martin 81b] have been considered. It has been verified on these applications that the general method described in [Martin 81a] and [Martin 81b] for mapping a concurrent computation on a parallel machine leads, in most cases, to a close to optimal spreading of the concurrent activities on the different processing elements.

6.1.2 Concurrent Algorithms as Space-time Recursion Equations

Marina Chen (Advisor: Carver A. Mead)

An early phase of this work was supported under this contract. Currently supported by the System Development Foundation and an IBM fellowship.

It is clear from existence proofs of such innovative designs as systolic arrays [Kung, Leiserson 80], tree machine algorithms [Browning 80], computational arrays [Johnsson et. al. 81], wavefront arrays [Kung 80], etc. that vast performance improvements can be achieved if the design of so-called "high-level" algorithms is released from the one dimensional world of a sequential process, and the cost of communications in space as well as cost of computation in time is taken into consideration [Sutherland, Mead 77]. While this higher dimensional design space provides a great playground for innovative algorithm design, it also introduces pitfalls unapprehended by those accustomed to the world of a single sequential process. Verification of algorithms becomes much more crucial in system designs because debugging concurrent programs can very easily become an exponentially complicated task in this rich space. The real difficulty lies in the high degree of complexity of concurrent systems. The well-known hierarchical approach can be used to manage the design complexity for such systems. A system is broken down into successive levels of sub-systems until each is of a manageable complexity. The effectiveness of this approach relies on two basic tools: A design and verification methodology for each level and an abstraction mechanism to go from one level to the next. The latter is crucially important, for without it the consistency of the whole system

is imperiled.

In the paper [Chen, Mead 82], we describe a methodology and a single notation for the specification and verification of synchronous and self-timed concurrent systems ranging from the level of transistors to communicating processes. The uniform treatment of these systems results in a powerful abstraction mechanism which allows management of system complexity.

Traditionally, due to the assumption that the cost of accessing variables in memory is the same regardless of their locations, sequential algorithms ignore the spatial relationships of variables. In addition, the steps of a computation have not been explicitly expressed as a function of time, but are rather implied by programming constructs. Languages that cannot express the spatial relationships of variables cannot take into account the most important aspect in the design of a concurrent algorithm, i.e. ensuring locality of communications, taking advantage of the interplay of variables in space (in practice up to 3 dimensions) to achieve higher performance. The implicit "time" causes programming languages to suffer either from not being able to abstract the history of computation (e.g. in applicative and data-flow languages [Kahn 74], [Backus 78]), or not being able to abstract computation in a clean functional form (e.g. in assignment-based languages). Here we choose to make "time" an explicit parameter of computation. We call our representation of computation a "Space-time Algorithm".

In [Chen xxxx:TR:82], CRYSTAL (Concurrent Representation of Your Space Time ALgorithm), a notation for concurrent programming is proposed. The fixed-point approach [Scott, Strachey 71] is used for characterizing the semantics. Within this framework, a program is expressed as a set of systems of recursion equations. Unknowns of the equations are data expressed as functions from the space-time domain to the value domain. For a deterministic concurrent system, such as a systolic array, a single system of equations results, and the semantics of such a system is defined as the least solution of the equations. The semantics of concurrent systems in general can be characterized as the corresponding set of solutions of the set of systems of equations. Various inductive techniques (see for example [Manna 74] used in verifying recursive programs can be directly applied in verifying space-time algorithms and proving their properties. We have applied this framework to transistor circuits, logic gates, arithmetic units, sequential processors, ensembles of large numbers of communicating processes such as the synchronous and self-timed version of the matrix multiplication on systolic arrays [Kung, Leiserson 80], and Tree Machine Algorithms [Browning 80]. In the systolic type of computations, the notion of wavefront is especially important. We define the "phase" of a computation wave in a way that is analogous to the wave in physical world. The set of all possible "phases" can be formalized as a well-founded set, upon which the inductive proof is based.

6.1.3 A Characterization of Deadlock Free Resource Contentions

Marina Chen (Advisor: Carver A. Mead)

Suppose we have a system consisting of some finite number of concurrent processes. The processes are said to be synchronized whenever the progress of some process may have to be delayed because of conditions caused by the other processes in the collection. Whenever processes are synchronized there exists the possibility that they may maneuver themselves into a deadlock [Habermann 69]. A deadlock is a state in which there is at least one non-terminated process indefinitely delayed, i.e., such that no state can be reached from which the process can make further progress.

An important class of deadlock problems arises from conflicts over shared resources, each of which can be used by only one process at a time. For example, one process while using resource A may require resource B. However resource B may already be in use by a second process which, in turn, requires resource A in order to proceed. Both processes are therefore stalemated and will remain so indefinitely: they have reached a deadlock.

The problem of deadlock arising from shared resources was first noticed in operating systems having quasi-parallel processes [Dijkstra 68]. These processes, although logically parallel, are sequentialized when implemented by a traditional sequential machine. Thus a natural solution for preventing deadlock is by a global scheduler which controls the sequence of resource assignments. A well-known scheme for deadlock prevention is the "banker's algorithm" [Dijkstra 68]. In this scheme the maximum amount of each resource each process ever requires is given. There is one scheduler for all the resources. It keeps track of which resources each process is using, i.e., the "state" of the system. The banker's algorithm entails a method of computing the "safety" of states. Each time a process requires a resource, the scheduler will allow the use of that resource only if the ensuing state is still safe. The banker's algorithm is an example of a global scheduler.

In the paper [Chen, Rem, Graham 4684:TR:82] we look at resource sharing without a global scheduler. In fact, the only scheduling we allow is the delaying of a process requiring a resource already in use. The acquisition of resources is concurrent and asynchronous, and no global information about it is available. A process is always allowed to free a resource; the resource returned is then available for use again. Such "mutual exclusion scheduling" can, for example, be realized with semaphores [Dijkstra 68]. Each resource has a binary semaphore and an arbitration mechanism. We do not assume any particular arbitration scheme. This type of "minimal scheduling" is important, for example, in VLSI systems [Mead, Conway 80], in which the requirement to have a global scheduler would seriously degrade its computing potential.

We establish necessary and sufficient conditions for collections of synchronized processes to be free from deadlock and prove a general theorem which characterizes those collections which have deadlock potential. We examine the computational complexity for testing the deadlock condition and derive a polynomial time algorithm for the problem. We also discuss the special case that all resources are of different types.

6.1.4 A Structured Petri Net Approach to Concurrency

Young-il Choo (Advisor: Jim Kajiya)

6.1.4.1 Hierarchical Nets

Liveness and safeness are two key properties Petri nets should have when they are used to model asynchronous systems. The analysis of liveness and safeness for general Petri nets, though shown to be decidable by [Mayr 81], is still computationally expensive ([Lipton 76]). In this paper an hierarchical approach is taken: a class of Petri nets is recursively defined starting with simple, live and safe structures, becoming progressively more complex using net transformations designed to preserve liveness and safeness.

Using simple net transformations, nice nets, which are live and safe, are defined. Their behavior is too restrictive for modeling non-trivial systems, so the mutual exclusion and the repetition constructs are added to get m-r-nets.

Since the use of mutual exclusions can cause deadlock, and the use of repetitions can cause loss of safeness, restrictions for their use are given. Using m-r-nets as the building blocks, hierarchical nets are defined. When the mutual exclusion and repetition constructs are allowed between hierarchical nets, distributed hierarchical nets are obtained. Distributed hierarchical nets are used to solve the Smokers problem by Patil and the Dining Philosophers problem. These solutions are compared with Lauer and Campbell's [Lauer, Campell 75] path expressions'.

General net transformations not preserving liveness or safeness, and a notion of duality are presented, and their effect on Petri net behavior is considered.

6.1.4.2 The Infinite Shuffle

In studying the behavior of Petri nets, the most interesting cases are when the nets have non-terminating behavior. For example, let A and B be Petri nets generating a's and b's respectively in a cycle. By the definition of Petri nets, the composite behavior of A parallel with B is the set of all fair merges. The resulting behaviors are infinite and they cannot be approximated using finite initial behaviors.

We have an abstract definition of the infinite shuffle, but we would like to relate it to a more operational one that is closer to the underlying implementation using probabilities.

GOAL: Is there any way to assign a probability to the nth occurance of a, say, as a function of the prefix generated up to then, so that the probability of generating an unfair merge is zero, but a fair merge is non-zero?

PROBLEMS: There are an uncountable number of fair strings and countable number of unfair ones. The sum of the probability of all fair strings should be one. The unfair ones zero. If each letter has probability less than one, then any infinite sequence has probability zero. What numbers can be assigned to the probabilities of the fair strings so that it makes sense? There are infinite series (countable) which sum to finite numbers, but are there uncountable sets which also sum to finite

numbers?

DIRECTION: One possible direction is to look at the non-standard real numbers first constructed by A. Robinson [Robinson 66]. We want to consider probability with non-standard measures ([Bernstien, Wattenburg 69]). Using infinitesimal numbers, the discontinuity between the finite and infinite shuffle may be bridged.

6.2 Functional Programming Languages

6.2.1 Experiments with data abstractions

Jim Kajiya

The promise of VLSI to solve major computation problems is hampered by our ability to program such new architectures. Even today we see that it is often software, not hardware, which is the limiting factor in application of computers for military purposes. The increased capability of VLSI hardware may go untapped if it is designed without regard to its programmability. We are conducting programming language experiments which seek to translate the projected increase in compute power offered by VLSI into vastly increased programmer productivity.

There are three programming language experiments which vary according to risk and potential payoff. The first is called Fith, an extremely small language which explores applicability of data abstraction and late binding to systems programming. The second is called ATP, which attempts to include data abstraction into a mature functional language. The third, 81, attempts to fuse functional and logic programming.

Fith is a kernel language which is extremely easy to implement [Kajiya xxxx:TR:82]. Syntactically, it represents most intermediate languages of compilers and is a very close relative to FORTH. Semantically, it is virtually identical to Smalltalk. The question that this experiment seeks to answer is: Can late binding data abstraction languages be made efficient enough for systems programming? The well known advantages of Smalltalk and its descendants (PIE, flavors, actors) have been denied to systems programmers because typical runtime structures are very inefficient. Fith recovers much of this efficiency by certain strategies to drastically minimize the need for storage management, the principal mechanism behind the inefficiency of the aforementioned languages. We are in the final stages of implementing Fith, these include a compiler, interpreter, runtime symbolic debugger, process and context managers, and garbage collectors. We will soon attempt to gain extensive experience with the language in order to test its suitability as a systems programming language.

The second language experiment is called ATP. In this experiment, the functional language known as array theory -- which is an extension of APL allowing recursive arrays -- is combined with data abstraction [Kajiya 82c]. This combination creates a synergy between the functional and data abstraction approaches engendering a new style of programming. In this style, operators are used to define the pattern of message passing among objects arrayed in data structures. For example, to send a dyadic message f between the objects in a vector A B C D, it is sufficient to write f/A B C D to give the result AfBfCfD; rather than

to have to write a loop or recursive function such as:

```
message reducef(a) begin
        x:=a[1];
        for i:=2 to 4 do
                x:=x f a[i];
        end
```

which steps through a vector accumulating the result of concatenating the message to the initial object. We are very excited by this language: it appears to increase programmer productivity by more than an order of magnitude for signal processing and graphics applications. We are still in the planning stages for implementing this language.

The third language experiment is called 81. It is the most speculative but also promises the highest payoff. With this language, the notions of functional and logic programming are combined. It appears that functional languages have great power for programming in the small but do less well for large projects. On the other hand, logic programming is extremely powerful for programming in the large but is weak for small applications. A combination of these two languages would have each fill in the others weaknesses. It turns out that the combination of these two programming styles is far from trivial. We were able to easily write programs in such a style but had absolutely no clue how to execute such programs! The key to the implementation has turned out to be an idea in mathematical logic we have called meta-recursive model theory [Kajiya xxxx:TR:82], viz. studying models which are identical to the language used to speak about it. It appears that these models are very natural for functional languages. Indeed, the notions are implicit within ideas presented by (Backus 1981) and in the interpretations of the lambda calculus and LISP which are separate from the Scott lattice model. In particular, the semantics of the LISP quotation conventions is amenable to formal treatment using such models. We are currently exploring the notion of logical inference in such models. Inference must be understood before such notions as soundness and completeness may be defined, which in turn are necessary before much work on implementing the deduction procedure which effects the programming language processor.

6.2.2 Type Inference of Late Binding Languages

Eric Holstege (Advisor: Jim Kajiya)

The following project performs a procedure known as type inference on the popular late binding language Smalltalk. Type inference deduces appropriate bookkeeping information key to efficient execution of late binding languages. This class of late binding languages (which include Smalltalk, LISP, actors, etc.) has been found to offer significant advantages in programmer productivity in an interactive environment. In other object oriented languages the user is required to explicitly maintain and supply this bookkeeping information. By deducing such information this compiler produces code suitable for execution on either sequential machines or highly concurrent machines such as Lang's, [Lang 5014:TR:82].

Object-oriented languages are desirable not only for having a data

abstraction mechanism corresponding to PASCAL-style records, but also a data encapsulation giving greater control over the manipulation of data. An object-oriented language class specifies the organization of data and also the operations which are to be allowed on it. This protection of data from arbitrary corruption gives a greater security and error avoidance. In addition, it has been found at Caltech from the heavy use of SIMULA, and more recently MAINSAIL, that the object-oriented style is conceptually easy to program in, providing a useful framework for the subdivision of large problems into manageable pieces.

The second major area focused on is declarationlessness. In most widely used general purpose languages, the programmer must declare the types of all the variables he uses. These variables then remain of the specified type throughout the execution of the program. This allows the compiler to produce efficient code and to identify errors whose detection must otherwise be deferred until runtime; however, it sacrifices a good deal of the generality which is possible with less stringent variable binding schemes. On the other hand, languages which don't require declarations, such as SNOBOL and LISP, allow considerable generality by virtue of their extremely late binding, but thereby sacrifice efficiency.

Smalltalk is an example of the latter category. Variables can be of any data type, or "class", including user defined ones, and can change types merely by appearing on the left side of an assignment statement. Associated with each class is a set of operations allowed on variables of that class, which completely defines the legal manipulations of the data.

Unfortunately, the twin features of declarationlessness and object-orientedness, while of great benefit in increasing programmer productivity and program reliability, suffer heavily from the point of view of runtime efficiency.

In this effort we are investigating ways to obtain the undeniable advantages of declarationlessness and object-orientedness, without sacrificing runtime efficiency. More specifically, the goal is to build a compiler for a dialect of Smalltalk for the VAX under UNIX (Berkeley 4.1bsd), which incorporates data flow type-inference algorithms enabling it to produce executable programs of an efficiency comparable to that of programs produced by compilers for more traditional but less powerful languages.

6.2.3 An APL compiler

Howard Derby (Advisor: Jim Kajiya)

APL is a programming language with extreme expressive power and inherently parallel semantics. Such characteristics make the language and its descendants prime candidates for implementation on concurrent VLSI systems. This study attempts to overcome many of the problems encountered in compiling such a language.

APL is about 20 years old, but most implementations still interpret it. Hewlett-Packard has an implementation which compiles small sections of APL programs as it goes, but there are no implementations which "compile". This is primarily due to the completely dynamic typing used

by APL, which makes it impossible to simply generate code on a statement by statement basis. This is unfortunate, because APL's ability to manipulate arrays make it a very powerful language for signal processing which require such large amounts of computing that compiling is almost mandatory.

Our research for the past year has centered on production of a prototype APL compiler. It is being written in the language PROLOG. Because of it's unique ability to backtrack and to unify trees, the use of PROLOG greatly simplifies the programming effort.

The compiler is based upon a system of rules that define the behavior of the various APL functions and operators. The result of an APL operation has 4 attributes: type, rank (number of dimensions) , shape (size along each dimension), and value at each index. For each operation, there are rules that specify each of these attributes in terms of its arguments. For the shape (which is a vector) , and value (which may have any number of dimensions) , the rules specify how to access an element of the array from its subscript.

From these rules, the compiler is able to generate code for executing APL expressions. To generate efficient code it needs to know the type and rank of any APL variables that appear in the expression. To get this information, it uses the same rules in an interpretive mode to evaluate only those parts of the program that are needed.

Compiling APL is not as easy as it may seem. Complications are introduced by branches in control flow, loops, and user functions. Since code for an expressions makes assumptions about the variables it contains, machine code control flow cannot be simply modeled after APL control flow. Compiler sections are needed to keep track of the assumptions used and force new machine code to be generated when it is needed.

6.3 Theory of Computer Science

6.3.1 D-infinity as a Model of Omega-Order Intuitionistic Logic

Leonid Rudin (Advisor: Jim Kajiya)

Partially supported by a fellowship.

The main aim of this work is to formulate natural foundations for type-free illative lambda-calculus. Scott's D-infinity model for the pure lambda-calculus is extended to include a lattice algebraic model for propositional logic by proving the following theorem:

Let D-0 be a finite relatively pseudo-complemented lattice. Then D-infinity is a complete relatively pseudo-complemented lattice with the operation of relative pseudo-complementation given by

$$\text{N-th component of}(X \Rightarrow Y) = \operatorname{g.l.b}_{n=0}^{\inf} \{(Xn \Rightarrow Yn), \ldots \text{Psi}(n+1,n)(X(n+1) \Rightarrow Y(n+1)) \ldots\}.$$

This shows that the D-infinity model is amenable to an axiomatization of

intuitionistic propositional calculus. We have found that it is not possible to strengthen this result to Boolean logic due to the following result:

Theorem. The maximal Boolean subalgebra of D-infinity is contained in D(0,infinity).

Using the model provided by the above two theorems we are able to resolve the so-called classical paradoxes of illative combinatory logic (e.g. Curry's) by restricting the notion of application and abstraction to (lattice) continuous terms. All ordinary lambda-calculus terms are continuous. Based on this interpretation of implication we build an axiom system for illative logic. As in all model-based axiomatizations we are guaranteed consistency through the existence of a model. Because of the completeness of the Scott lattice, this theory may be extended to Omega-order predicate calculus.

I. Publications, Technical Reports and Internal Memorandas (ARPA)

Barton Antony F., "A Fault Tolerant Integrated Circuit Memory," 3761:TR:80, Ph.D. Thesis, Computer Science, Caltech, April 1980.

Browning, Sally A. and Charles L. Seitz, "Communication in a Tree Machine," Proceedings of the Second Caltech Conference on VLSI, January 1981.

Browning, Sally A., "Generating Padding Processors for Arbitrary Fanout Trees", 3827:DF:80, Computer Science, Caltech, July 1980.

Browning, Sally A., "The Tree Machine: A highly concurrent computing environment," 3760:TR:80, Ph.D. Thesis, Computer Science, Caltech, January 1980.

Bryant, Randal E., Jack B. Dennis, "Concurrent Programming", 5027:TR:82, Computer Science, Caltech. To appear in Operating Systems Engineering, M. Maekawa, ed., Springer Verlag, 1982.

Bryant, Randal E., Mike Shuster, Doug Whiting, "MOSSIM II: A Switch-Level Simulator for MOS LSI, User's Manual", 5033:TR:82, Computer Science, Caltech, 1982.

Bryant, R. E., "Switch-Level Modeling of MOS Digital Circuits", International Symposium on Circuits and Systems, IEEE, May, 1982.

Bryant, Randal E., "A Switch-Level Model of MOS Logic Circuits", Proceedings of the First International Conference on VLSI, VLSI 81, Academic Press, 1981, pp. 329-340.

Bryant, Randal E., "A Switch-Level Simulation Model for Integrated Logic Circuits", Laboratory for Computer Science, MIT, Technical Report MIT/LCS/TR-259, March 1981.

Carroll, Chris, "Hybrid Processing", 5034:TR:82, Ph.D. Thesis, Computer Science, Caltech.

Carroll, Chris R., "A Smart Memory Array Processor for Two-Layer Path Finding," Proceedings of the Second Caltech Conference on VLSI, January 1981.

Carroll, Chris R., "Hardware Path Finders," Computer Science, Caltech, internal document, September 1980.

Chen, Marina, Carver A. Mead, "Formal Specification of Concurrent Systems", 5042:TR:82, Computer Science, Caltech, October 1982.

Chen, Marina, Carver A. Mead, "On the Semantics of Space-Time Algorithms: A Summary", 5020:DF:82, Computer Science, Caltech.

Chen, Marina, Rem, Martin, and Graham, Ronald, "A Characterization of Deadlock Free Resource Contentions", 4684:TR:82, Computer Science, Caltech, January 1982.

Chen, Marina, "A Semantics for Systolic Arrays", 4635:DF:81, Computer

Science, Caltech, August 1981.

Chen, Marina, "HARMOS-A Notation for Designing Concurrent Systems", 3927:DF:80, Computer Science, Caltech, August 1980.

Choo, Young-il, "Concurrency Algebra: Towards an Algebraic Semantics of Petri Nets," 4085:DF:80, Computer Science, Caltech, December 1980.

DeBenedictis, Erik, "Testing and Structured Design", Proceedings of the IEEE International Test Conference, Philidelphia, PA, November 1982. Also, 4778:DF:82, Computer Science, Caltech, August 1982.

DeBenedictis, Erik, "An Embeddded Concurrent Language", 4781:DF:82, Computer Science, Caltech.

DeBenedictis, Erik, "Techniques for Testing I.C.", 4777:TR:82, Computer Science, Caltech.

DeBenedictis, Erik, "A Communications Operating System for the Homogeneous Machine", 4707:DF:82, Computer Science, Caltech, January 1982

DeBenedicitis, Erik, "Homogeneous Machine Technical Plan", 4705:DF:82, Computer Science, Caltech, January 1982.

DeBenedictis, Erik, "FIFI Test Instrument: Software Specification", 4686:DF:81, Computer Science, Caltech, November 1981.

DeBenedictis, Erik, "A Methodology for Describing Test Specifications", 4685:DF:81, Computer Science, Caltech, November 1981.

DeBenedictis, Erik, "FIFI Test System, Preliminary User's Manual," 4270:TR:81, Computer Science, Caltech, April 1981.

DeBenedictis, Erik, "A Preliminary Report on the Caltech ARPA tester project," 4061:TR:80, MS Thesis, Computer Science, Caltech, April 1980.

DeBenedictis, Erik, "Justification for a Tree Machine", 3751:DF:80, Computer Science, Caltech, May 1980.

Holstege, Eric, "Type Inference in a Declarationless, Object-Oriented Language", 5035:TR:82, MS Thesis, Computer Science, Caltech.

Johnsson, Lennart and Danny Cohen, "A Formal Derivation of Array Implementations of FFT Algorithms", USC Workshop on VLSI and Modern Signal Processing", November 1 - 3, 1982. (Also available as 5043:TM:82, Computer Science, Caltech)

Johnsson, Lennart, "Concurrent Algorithms for the Conjugate Gradient Method", 5040:TR:82, Computer Science, Caltech, September 1982.

Johnsson, Lennart, "VLSI Algorithms for Doolittle's, Crout's and Cholesky's Methods", IEEE International Conference on Computers and Circuits, ICCC82, September 28 - October 1, 1982. (Also available as 5030:TM:82, Computer Science, Caltech)

Johnsson, Lennart, "Pipelined Linear Equation Solvers and VLSI", Microelectronics 1982, May 12-14, 1982. (Also available as 5003:TM:82, Computer Science, Caltech)

Johnsson, Lennart, "A Computational Array for the QR-method", The MIT Conference on Advanced Research in VLSI, January 25-27, 1982. (Also available as 5019:TM:82, Computer Science, Caltech)

Johnsson, Lennart and Danny Cohen, "A Mathematical Approach to Modelling the Flow of Data and Control in Computational Networks", The CMU Conference on VLSI Systems and Computations, Pittsburgh, October 19-21, 1981.

Johnsson, Lennart, "Computational Arrays for Band Matrix Equations", 4287:TR:81, Computer Science, Caltech, May 1981.

Johnsson, Lennart and Danny Cohen, "Computational Arrays for the Discrete Fourier Transform," Proceedings of the Twenty-Second Computer Society International Conference, COMPCON 81, February 1981. (Also available as 4168:TR:81, Computer Science, Caltech)

Johnsson, Lennart, Uri Weiser, Danny Cohen and Alan L. Davis, "Towards a Formal Treatment of VLSI Arrays," Proceedings of the Second Caltech Conference on VLSI, January 1981. (Also available as 4191:TR:81, Computer Science, Caltech)

Johnsson, Lennart, "A Note on Householder's Method, Sparse Matrices and Concurrency," 4089:DF:80, Computer Science, Caltech, December 1980.

Johnsson, Lennart, "Gaussian Elimination on Sparse Matrices and Concurrency," 4087:DF:80, Computer Science, Caltech, December 1980.

Kajiya, J., "Ray Tracing Parametric Patches", SIGGRAPH-82, Computer Graphics v.16, no.3 July 1982, pp. 245-254. (Also available as 5017:TM:82, Computer Science, Caltech)

Kajiya, and Ullner, "Filtering High Quality Text for Display on Raster Scan Devices", SIGGRAPH81, Computer Graphics v.15, no.3, August 1981, pp. 7-15. (Also available as 5018:TM:82, Computer Science, Caltech)

Kajiya (1981) "Generic functions by nonstandard name scoping in APL", APL81 Conf. Proc., Quote Quad v.12, no. 1, September 1981, pp.172-179.

Kajiya (1982) "Initial report on the Fith experiment: a language incorporating late binding and abstract datatypes intended for systems programming" Internal Caltech Document.

Kajiya (1982) "Meta-Recursive Model theory" Internal Caltech Doc.

Kajiya (1983) "Designing and Implementing an Array Theory Incorporating Abstract Datatypes" to appear in proceedings of APL83.

Kajiya (1983) "Rendering stochastically defined surfaces." Submitted to SIGGRAPH83.

Kajiya and Gabriel (1983) "Splining in curved manifolds" Submitted to SIGGRAPH83

Kingsley, Chris, "Earl: An Integrated Circuit Design Language", 5021:TR:82, MS Thesis, Computer Science, Caltech, June 1982.

Lang, Dick, "The Extension of Object-Oriented Languages to a

Homogeneous, Concurrent Architecture", 5014:TR:82, Ph.D. Thesis, Computer Science, Caltech, May 1982.

Lang, Dick, "Concurrent, Asynchronous Garbage Collection Among Cooperating Processors", 4724:TR:82, Computer Science, Caltech, February 1982.

Lang, Dick, "A Distributed Class Object Processing Architecture," 4199:DF:81, Computer Science, Caltech, February 1981.

Lewis, Robert, "Switching Dynamics", 4675:TR:81, Computer Science, Caltech, October, 1981.

Li, Peggy, "The Tree Machine Operating System", 4618:TM:81, Computer Science, Caltech, July 1981.

Li, Peggy, "The Serial Logarithm Machine," 4517:TR:81, MS Thesis, Computer Science, Caltech, November 1980.

Lien, Sheue-Ling, "Towards a Theorem Proving Architecture", 4653:TR:81, MS Thesis, Computer Science, Caltech, July 1981.

Locanthi, Bart, "The Homogeneous Machine", 3759:TR:80, Ph.D. Thesis, Computer Science, Caltech, January 1980.

Martin, Alain J., "A Distributed Implementation Method for Parallel Programming," Information Processing 80, S. H. Lavington (ed.), pp. 309-314., Oct. 1980. (Also available as 5045:TM:82, Computer Science, Caltech)

Martin, Alain J., "An Axiomatic Definition of Synchronization Primitives," Acta Informatica 16, 219-235, 1981. (Also available as 5046:TM:82, Computer Science, Caltech)

Martin, Alain J., "The Torus: An Exercise in Constructing a Processing Surface," Proceedings of the 2nd Caltech Conference on VLSI, January 1981. (Also available as 5047:TM:82, Computer Science, Caltech)

Martin, Alin J., "Distributed Mutual Exclusion Algorithms," AJM 31, Sept. 1982 (submitted for publication)

Martin, Alain J., "Communication in Distributed programming," Caltech internal document, Sept. 1982

Mead, Carver and Rem, Martin, "Minimum Propagation Delays in VLSI", Proceedings of the Second Caltech Conference on VLSI, January 19-21, 1981, also published as 4601:TM:81, Computer Science, Caltech, 1981.

Rem, Martin and Carver Mead, "A Notation for Designing Restoring Logic Circuitry in CMOS," Proceedings of the Second Caltech Conference on VLSI, January 19-21, 1981, also published as 4600:TM:81, Computer Science, Caltech 1981.

Rem, Martin, "Communication in a Binary Tree, I and II, Some observations on a tree mapping problem," internal documents, Computer Science, Caltech, June and July 1980.

Rudin, Leonid and J.T. Kajiya, D-infinity As a Model of Omega-order

Intuitionistic Logic,Abstracts of Logic Colloquium `82 & 1982 European Summer Meeting of the ASL (Florence, August 23-28 1982).

Rudin, Leonid, "Lambda-Logic", 4521:TR:81, MS Thesis, Computer Science, Caltech, May 1981.

Schuster, Mike, Randal E. Bryant, "MOSSIM II: A Switch-Level Simulator for MOS LSI, User's Manual, 5033:TR:82, Computer Science, Caltech.

Seitz, Charles L, et al, "Proposed instruction set for the tree machine processor," internal document, Computer Science, Caltech, July 1980.

Whelan, Dan, "A Rectangular Area Filling Display System Architecture", to be presented at SIGGRAPH-82, July, 1982. Also to appear in Computer Grahics.

Whelan, Dan, "A Versatile Ethernet Interface", 4654:TR:82, Computer Science, Caltech, July 1981.

Whelan, Dan and Ray Eskenazi, "An Inexpensive Multibus Color Frame Buffer", 4334:TR:81, Computer Science, Caltech, June, 1981.

Whiting, Doug, "A Self-Timed Chip Set For Multiprocessor Communication", 5000:TR:82, MS Thesis, Caltech.

Internal documents (DF) may be obtained only from the authors. Technical Reports (TR) and Technical Memoranda (TM) can be obtained from the Computer Science Librarian, Room 256-80, California Institute of Technology, Pasadena, California 91125. Please identify yourself to the librarian as a member of the ARPA community!

REFERENCES

[Backus 78]
Backus, J.
Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs.
CACM 21(8):613-641, August, 1978.

[Bernstien, Wattenburg 69]
Bernstien, A. R., Wattenburg F.
Nonstandard Measure Theory.
In Luxemburg, W. A. J., editor, Application of Model Theory to Algebra, Analysis, and Probability, pages 171-185. Holt, Reinhart, and Winston, 1969.

[Browning, Seitz 81]
Browning, S. A., Seitz, C., L.
Communication in a Tree Machine.
In Proceedings of the Second Caltech Conference on VLSI. Computer Science, Caltech, 1981.

[Browning 80]
Browning, Sally A.
The Tree Machine: A Highly Concurrent Computing Environment.
Technical Report 3760, Caltech Computer Science Department, January, 1980.

[Chen, Mead 82]
Chen, M. C. and Mead C. A.
Concurrent Algorithms as Space-time Recursion Equations.
In Proceedings of USC Workshop on VLSI and Modern Signal Processing. Computer Science, University of Southern California, November, 1982.

[Clark 67]
Clark, W. A.
Macromodular Computer Systems.
In AFIPS Conference Proceedings. AFIPS, 1967.

[DeBenedicitis 82]
DeBenedicitis, E.
Testing and Structured Design.
In Proceedings of the IEEE International Test Conference. IEEE, 1982.

[Dijkstra 68]
Dijkstra, E. W.
Cooperting Sequential Processes.
In Genuys, F., editor, Programming Languages, pages 43-112. Academic Press, 1968.

[Habermann 69]
Habermann, A. N.
Prevention of System Deadlock.
Communications of the ACM 12(7):373-377, 1969.

[Johnsson et. al. 81]
Johnsson, S. Lennart, Weiser, U., Cohen, D., and Davis, A.
Towards a Formal Treatment of VLSI Arrays.
In Proceedings of the Second Caltech Conference on VLSI, pages . Caltech Computer Science Department, January, 1981.

[Johnsson, Cohen 82a]
Johnsson, S. L., Cohen, D.
An Algebraic Description of Array Implementations of FFT Algorithms.
In 20th Allerton Conference on Communication, Control, and Computing. Electrical Engineering, University of Illinois, Urbana/Champaign, 1982.

[Johnsson, Cohen 82b]
Johnsson, S. L., Cohen, D.
A Formal Description of Array Implementations of FFT Algorithms.
In Proceedings of the USC Workshop on VLSI and Modern Signal Processing. Computer Science, University of Southern California, 1982.

[Johnsson 82a]
Johnsson S.L.
VLSI Algorithms for Doolittle's, Crout's and Cholesky's Methods.
In International Conference on Circuits and Computers 1982, ICCC82, pages 372-377. IEEE, Computer Society, September, 1982.

[Johnsson 82b]
Johnsson L.
A Computational Array for the QR-method.
In Paul Penfield, Jr., editor, Proceeedings, Conferences on Advanced Research in VLSI, pages 123-129. Artech House, January, 1982.

[Johnsson 82c]
Johnsson S. L.
Pipelined Linear Equation Solvers and VLSI.
In Microelectronics '82, pages 42-46. Institution of Electrical Engineers, Australia, May, 1982.

[Kahn 74]
Kahn, G.
The Semantics of a Simple Language for Parallel Programming.
In Proceedings, IFIP Congress. IFIP, 1974.

[Kajiya 82a]
Kajiya, J.
Ray Tracing Parametric Patches.
Computer Graphics 16(3):245-254, July, 1982.

[Kajiya 82b]
Kajiya, J.
Rendering Stochastically Defined Surfaces.
Submitted to SIGGRAPH83.

[Kajiya 82c]
Kajiya, J.
Designing and Implementing an Array Theory Incorporating Abstract Data Types.
To appear in Proceedings of APL83.

[Kung, Leiserson 80]
Kung, H.T. and Leiserson, Charles E.
Algorithms for VLSI Processor Arrays.
In Introduction to VLSI Systems, pages 271-294. Addison-Wesley, 1980.
Mead, Carver A. and Conway, Lynn A.

[Kung 80]
Kung, S.Y.
VLSI Matrix Computation Array Processor.
In MIT Conference on Advanced Research in Integrated Circuits, pages . MIT, February, 1980.

[Lauer, Campell 75]
Lauer, P. E., Campell R. H.
Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes.
Acta Informatica 5:297-332, 1975.

[Lipton 76]
Lipton, R.
The Reachability Problem Requires Exponential Space.
Technical Report Research Report 62, Department of Computer Science, Yale University, January, 1976.

[Manna 74]
Manna, Z.
Mathematical Theory of Computation.
McGraw-Hill, 1974.

[Martin 81a]
Martin, A.
A Axiomatic Definition of Synchronization Primitives.
Acta Informatica 16:219-235, 1981.

[Martin 81b]
Martin, A.
The Torus: An Exercise in Constructing a Processing Surrface.
In 2nd CALTECH Conference on VLSI. Computer Science, California Institute of Technology, January, 1981.

[Martin 82a]
Martin, A.
Communication in Distributed Programming.
Technical Report, California Institute of Technology, September, 1982.

[Martin 82b]
Martin, A.
Distributed Mutual Exclusion Algorithms.
Submitted for Publication, September 1982.

[Mayr 81]
Mayr, E. W.
An Algorithm for the General Petri Net Reachability Problem.
In Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pages 238 - 246. ACM, May, 1981.

[Mead, Conway 80]
Mead, C. A., Conway L. A.
Introduction to VLSI Systems.
Addison Wesley, 1980, .

[Mead, Rem 81]
Mead, C. A., Rem, M.
Minimum Propagation Delays in VLSI.
In Proceedings of the Second Caltech Conference on VLSI. Computer Science, Caltech, 1981.

[Robinson 66]
Robinson, A.
Non-Standard Analysis (Studies in Logic and the Foundations of Mathematics.
North-Holland, 1966.

[Scott, Strachey 71]
Scott, D., Strachey, C.
Toward a Mathematical Semantics for Computer Languages.
In Fox, J., editor, Symposium on Computers and Automata, . Wiley-Interscience, 1971.

[Seitz 70]
Seitz, C. L.
Asynchronous Machines Exhibiting Concurrency.
In Proceedings of the Project MAC Conference on Concurrent Systems and Parallel Computation. AFIPS, 1970.

[Seitz 80]
Seitz, C. L.
System Timing.
In Introduction to VLSI Systems, chapter 7. Addison-Wesley, 1980.

[Seitz 82a]
Seitz, C. L.
Ensemble Architectures for VLSI - A Survey and Taxonomy.
In Proceedings, Conference on Advanced Research in VLSI, pages 130 - 135. Artech House, 1982.

[Seitz 82b]
Seitz, C. L.
The Education of VLSI Designers at the University Level.
In Compcon 82, pages 106 - 108. IEEE Computer Society, 1982.

[Sutherland, Mead 77]
Sutherland Ivan, Mead Carver A.
Microelectronics and Computer Science.
Scientific American 237(3):210-229, September, 1977.

[Turner 77]
Turner, D. A.
A New Implementation Technique for Applicative Languages.
In Software Practice & Experience, pages 31 - 49. John Wiley & Sons, 1977.

[Whelan 82]
Whelan, D.
A Rectangular Area Filling Display System Architecture.
In SIGGRAPH-82. ACM, 1982.