DTIC FILE COPY

4

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

AD-A208 070

CRITICAL PROBLEMS IN VERY LARGE SCALE COMPUTER SYSTEMS

Semiannual Technical Report for the period October 1, 1988 to March 31, 1989

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Principal Investigators:	Paul Penfield, Jr.	(617) 253-2506
	Anant Agarwal	(617) 253-1448
	William J. Dally	(617) 253-6043
	Srinivas Devadas	(617) 253-0454
	Thomas F. Knight, Jr.	(617) 253-7807
	F. Thomson Leighton	(617) 253-3662
	Charles E. Leiserson	(617) 253-5833
	Jacob K. White	(617) 253-2543
	John L. Wyatt, Jr.	(617) 253-6718

This research was sponsored by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under Contract No. N00014-87-K-0825.

[illegible] 4

TABLE OF CONTENTS

* S. Devadas, H.-K. T. Ma, and A. R. Newton, "Easily Testable PLA-based Finite State Machines," to appear in *FTCS-19*, June 1989. Also to appear as MIT VLSI Memo No. 89-514, March 1989.

* A. Agarwal and A. Gupta, "Temporal, Processor, and Spatial Locality in Multiprocessor Memory References," MIT VLSI Memo No. 89-512, March 1989.

* S. Devadas, "Approaches to Multi-Level Sequential Logic Synthesis," to appear in *Proceedings of 26th Design Automation Conference*, Las Vegas, Nevada, June 1989.

* Ronald I. Greenberg, "Area-Universal Networks," extended abstract.

* A. T. Ishii, "*A Digital Model for Level-Clocked Circuitry*, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, August 1988.

* J. A. S. Fiske, "*A Reconfigurable Arithmetic Processor*," M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, December 16, 1988.

---

* Abstract only. Complete version available from Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; telephone (617) 253-8138.

RESEARCH OVERVIEW

The research vehicle for this contract is the largest possible computer that could be conceived for the mid to late 1990s. The technical challenges of such a machine serve as the guiding stimulus for the research carried out and reported here.

We imagine this machine to occupy a 14-story building, to cost upwards of $1,000,000,000, and to be so colossal that the nation could only afford one or two of them. The available chip technology and machine size are consistent with a million billion FLOPS (that's 10 to the 15th) and a million billion Bytes of memory. It will dissipate 50 megawatts of power using CMOS technology. Communication across the machine will be much slower than computation at a node. The architecture, software, interconnect technology, packaging, and operating system are unknown.

This investigation deals with hardware technology, software techniques, programming algorithms, communications, processing elements, and applications. The study will determine the plausibility (not feasibility) of such a machine. Progress in these various areas are highlighted in the individual sections below.

Keywords: circuits, systems software, communications topology, routing algorithms, (KR)

CIRCUITS

Our work over the past six months has been driven by two central themes: (1) the realization that with proper architectural support, we need not arbitrarily choose a single programming model to support on a parallel processor, and (2) an investigation of the deep relationship between constructing reliable, distributed, parallel memory systems and more conventional software database technology.

The support of multiple programming models (shared memory, dataflow, functional, message passing, systolic, data level parallelism etc.) on a single architecture appears now to be not only possible, but inevitable. With modest adjustments to processor, cache controllers, and interconnection technology, the requirement that each new parallel programming model demands a new system level design appears to be gone. In the processor, the main requirements are fast context switch and fast message dispatch and composition. In the cache controller, the requirement is for support of one of the newer message oriented cache coherency protocols. In the switch, the requirement is for extremely low latency, reliable communications.

Our approach has been to attack the interconnection issues first, on the assumption that they were least likely to be addressed carefully by others, and that they were (or could be made) simpler and easier to get right than the more complex designs for cache and processor. The Transit communication network is the outgrowth of this work.

Transit* is a small scale prototype network designed with fault tolerance and low latency as key design goals. Henry Minsky and Andre DeJon are working with me in the development of a custom VLSI component for this switch. The goal is to provide a 100 megabyte/second channel with 40 ns. latency between pairs of up to 256 processor ports in a reliable, fault tolerant design.

Packaging is key, and we are developing a liquid cooled three-dimensional package, based on the button board concept, which provides approximately equal wiring density in all three axes. The goal is to package an entire large scale multiprocessor into a solid cubical block of printed circuit board material approximately 18" on a side.

Innovative techniques are being used to transmit data between chips† modified by replacing the voltage controlled output switches by a binary weighted D/A resistive network.

The switch design is extremely simple, and involves no buffering, queuing, or combining, opting instead to concentrate on maximum speed of transmission. Fault tolerance is achieved with a combination of random route choice between equivalent paths and ethernet-style positive acknowledgement plus retry.

High expected success routing rates are achieved by the use of a 2x dilated omega network topology, which as Leighton and Koch show, has dramatically improved statistics over the conventional omega network.

Despite our detailed attention to the routing network design, preliminary thought is also being given to other portions of the design.

In the processor area, we are looking at techniques for speeding process switching, including the VLSI design of large multi-ported register files with built in backup copies, allowing single cycle register file switching.

---

* T. Knight, "Technologies for Low Latency Interconnection Networks," Symposium on Parallel Algorithms and Architectures, June, 1989

† T. Knight, "A Self-Terminating Low Voltage Swing CMOS Driver," *Journal of Solid State Circuits*, April, 1988.

In the cache area, we are investigating a variety of message based protocols cooperatively with Anant Agarwal's group. In addition, Neil Lackritz is working with me in attempting to understand the impact that a knowledge of hardware datatypes and their properties have on cache performance. Our hope is that we may gain cache performance over more conventional designs by relying on a combination of compiler directives and run time types of data to control the cache refill algorithms.

More fundamentally, we are beginning in earnest to examine the relationship between conventional software oriented database technology and the problem of maintaining consistent, replicated, distributed copies of main memory.

Alan Bawden, for example, is continuing his work on the utility and importance of side effects. His most important results to date include the first mathematical description of what a side effect is; he is now implementing a model of computation in which the costs (and benefits) of side effects are explicitly visible to the programmer.

Patrick Sobalvarro is extending our earlier work on dynamically checked side effects* by collecting traces of programs and implementing the coherency/concurrency techniques.

Bryan Butler, in collaboration with Draper Labs, is designing a novel fault tolerant main memory structure based on coding techniques, which is suitable for use in four way Byzantine fault tolerant architectures, uses half of the memory of alternative approaches, and dramatically (3 orders of magnitude) improves the predicted mean time to failure of the memory system.

* T. Knight, "An Architecture for Mostly Functional Languages," ACM Lisp and Functinal Programming Conference, August, 1986.

PROCESSING ELEMENTS

Our goal is to study the issues involved with and to develop technology for constructing building-sized multicomputers with 1997 technology. These machines will be of such a scale that their design will have to make the most efficient use of wires and energy.

The Named State Processor:

A computer the size of the ARC will require the ability to switch tasks rapidly to hide transmission latency without sacrificing single-thread performance. Peter Nuth and Bill Dally are working on an architecture for a named state processor that achieves this goal by explicitly binding names to all processor registers and interleaving tasks on a microcycle basis. This mechanism combines the advantages of multi-threading and multiple register sets for implementing fast context switches and procedure calls. It also provides a general synchronization mechanisms.

Naming state permits process switches to be performed in essentially zero time as no registers need be saved or restored. A process switch is performed by simply issuing an instruction fetch with a different process ID field. Unlike conventional multithreaded processors, this approach also permits the instructions of a single process to be pipelined, executing one per cycle, achieving good single thread performance.

Naming the processor state permits several processes to have instructions in the pipeline simultaneously. Pipeline bubbles due to data dependencies, memory latency, or interprocess communication are filled by advancing instructions from other processes.

The named state processor performs all synchronization through the use of presence tags on its state. Synchronization on register dependencies, memory references, and communication actions all use this single mechanism.

Since our last report we have refined the named state processor architecture and defined its interface to a multicomputer network. We are currently studying instruction scheduling policies (deciding which processes instructions get advanced when) and context cache management policies (deciding which processes state remains in active storage). A simulator for the processor is under construction.

Concurrent data abstractions:

Andrew Chien and Bill Dally are developing data abstraction tools that support the development of programs for large scale multicomputers. A language, concurrent data abstractions, is being defined that facilitates the specification of aggregates of cooperating objects. Concurrent data abstractions permit the relationships between objects to be defined textually rather than requiring that the objects connect up a pointer structure at run-time as is typically done. Common structures (e.g., combining trees) can be defined once and reused as required. The language also permits nesting of object aggregates and specialization of objects within the aggregate.

Database applications:

John Keen and Bill Dally are investigating the application of an ARC sized computer to database applications. The issues involved include data partitioning, methods for insuring stability and persistence, concurrency control, and efficient algorithms for search and update.

Fast Translation Method:

Bill Dally has developed a one-step translation method that implements paging on top of segmentation. This method translates a virtual address into a physical address, performing both the segmentation and paging translations, with a single TLB read and a short add. Previous methods performed this translation in two steps and required two TLB reads and a long add. Using the fast method, the fine-grain protection and relocation of segmentation combined with paging can be provided with delay and complexity comparable to paging-only systems. This method allows small segments, particularly important in object-oriented programming systems, to be managed efficiently.

Floating point optimization technique:

Bill Dally and Lucien Van Elsen have developed a technique, micro-optimization, for reducing the operation count and time required to perform numerical calculations. The method involves breaking floating-point operations into their constituent integer micro-operations and the optimizing and scheduling the resulting integer code. The method has been tested using a prototype expression compiler. We are now looking at extending the method to permit a compiler to perform automatic scaling of numbers. Where it is possible, this optimization would convert floating point expressions into integer expressions.

COMMUNICATIONS TOPOLOGY AND ROUTING ALGORITHMS

Charles Leiserson returned from a leave of absence at Thinking Machines Corporation January 1, 1989. He was an invited speaker at the 25th Anniversary Symposium for Project MAC at MIT, and at the Decennial Caltech VLSI Conference. He also served on the first program committee for the ACM Symposium on Parallel Algorithms and Architectures.

Charles Leiserson and Tom Cormen have been concentrating on finishing the textbook Introduction to Algorithms with Ronald Rivest. Besides offering a combined engineering and theoretical approach to computer algorithms, the book has several chapters devoted to parallel computing -- a novelty in the area. The book will be published jointly by MIT Press and McGraw-Hill by the end of 1989.

Shlomo Kipnis has been investigating parallel architectures and interconnection networks. He is trying to further explore the power of bussed interconnection schemes to route permutations. In addition, he is investigating the mesh and the hypercube interconnection schemes, and is looking into the problem of embedding one network in another network Recently, he has also studied the problem of range queries in computational geometry. Range queries is a fundamental problem in computational geometry with applications in computer graphics and database retrieval systems.

Marios Papaefthymiou joined the group in December 1988. He has been working with Charles Leiserson on algorithms for optimizing synchronous circuitry. Recently, he discovered an simple $O(E)$ algorithm for pipelining combinational circuitry to achieve a given clock period.

Jeff Fried is currently working on several problems related to the architecture and control algorithms needed for high performance communication networks. This work includes a study of the impact of synchrony on the performance of distributed algorithms, and design studies for a VLSI packet router chip. Fried completed his Master's thesis this semester. His thesis work involved the design of VLSI processors for use within the interconnection networks found in telecommunications, distributed computing, and parallel processing. He also completed a study of some of the modularity tradeoffs found in sparse circuit-switched interconnection networks. In the next year, Fried plans to continue his study of distributed algorithms and architectures to support them. He will also be considering a number of other problems relating to the design of switching nodes for use in broadband networks.

Cynthia Phillips continued her investigation of parallel graph contraction algorithms which lead to simple algorithms for connected components, biconnectivity, and spanning trees of graphs. She developed a simple contraction algorithm for general n-node graphs which runs in $O(\lg^2 n)$ time using $O(n)$ processors on an EREW PRAM. This algorithm is used in a contraction algorithm for bounded-degree graphs which runs in $O(\lg n + \lg^2 g)$ time where g is the maximum genus of any connected component. Also, with Stavros Zenios of the Wharton School at U. Pennsylvania, she began an investigation of parallel implementations of algorithms for network optimization problems. In particular, they investigated the behavior of known algorithms, embellished with heuristics, for the assignment problem and nonlinear network flow.

In the past six months, James K. Park has been collaborating with Alok Aggarwal and Dina Kravets on a number of problems relating to totally monotone arrays. Totally monotone arrays arise naturally in a wide variety of fields, including computational geometry, dynamic programming, string matching, and VLSI river routing. Park's work with Aggarwal centers on the problem of finding maximum entries in totally monotone arrays and applications of efficient sequential and parallel algorithms for this problem. Park's work with Kravets considers other comparison problems (such as sorting and computing order statistics) in the context of totally monotone arrays and applications of efficient solutions to these problems.

Alexander Ishii has completed his masters thesis[*], which describes his models for VLSI timing analysis. The model maps continuous data-domains, such as voltage, into discrete, or *digital*, data domains, while retaining a continuous notion of time. The majority of the thesis concentrates on developing lemmas and theorems that can serve as a set of "axioms" when analyzing algorithms based on the model. Key axioms include the fact that circuits in our model generate only well defined digital signals, and the fact that components in our model support and accurately handle the "undefined" values that electrical signals must take on when they make a transition between valid logic levels. In order to facilitate proofs for circuit properties, the class of *computational predicates* is defined. A circuit property can be proved by simply casting the property as a computational predicate.

Ishii has also been working with Bruce Maggs on a new VLSI design for a high-speed multi-port register file. Design goals include short cycle-time and single-cycle register window context changes. This research began as an advanced VLSI class project, under the supervision of Thomas Knight of the MIT Artificial Intelligence Laboratory.

Ishii has also been working with Ronald Greenberg and Alberto Sangiovanni-Vincentelli of Berkeley on a multi-layer channel router for VLSI circuits, called MulCh[†]. While based on the Chameleon system developed at Berkeley, MulCh incorporates the additional feature that nets may be routed entirely on a single interconnect layer (Chameleon requires the vertical and horizontal sections of a net be routed on different interconnect layers). When used on sample problems, MulCh shows significant improvements over Chameleon in area, total wire length, and via count.

Besides his work with Ishii and Sangiovanni-Vincentelli, Ronald Greenberg has been continuing work in two areas: channel routing for VLSI chips and area-universal networks for general-purpose parallel computation. With Miller Maley of Princeton, he has been developing techniques for efficiently determining minimum area routings for single-layer channels and switchboxes, which can be usefully incorporated into previous work on multi-layer channel routing. In the area of general-purpose parallel computation he has developed stronger and more general results about the ability of a machine built in a fixed amount of area to simulate other parallel machines.

Bruce Maggs has been working with Richard Koch, Tom Leighton, Satish Rao, and Arnold Rosenberg. They have been studying the ability of a host network to emulate a possibly larger guest network. An emulation is called "work-preserving" if the work (processor-time product) performed by the host is at most a constant factor larger than the work performed by the guest. A work-preserving emulation is important because it achieves optimal speedup over a sequential emulation of the guest. Many work-preserving emulations for particular networks have been discovered. For example, the N-node butterfly can emulate an $N \log N$ node shuffle-exchange graph and vice versa. On the other hand, a work-preserving emulation may not be possible unless the guest graph is much larger than the host. For example, a linear array cannot perform a work-preserving emulation of a butterfly unless the butterfly is exponentially larger than array. Worse yet, a work-preserving emulation may not exist. A butterfly cannot perform a work-preserving emulation of an expander graph. These positive and negative results provide a basis for comparing the relative power of different networks.

---

[*] Alexander T. Ishii, *A Digital Model for Level-Clocked Circuitry*, Master's thesis, Department of Electrical Engineering and Computer Science, MIT, August 1988.

[†] Ronald I. Greenberg, Alexander T. Ishii, and Alberto L. Sangiovanni-Vincentelli. MulCh: A multi-layer channel router using one, two, and three layer partitions. In *IEEE International Conference on Computer-Aided Design (ICCAD-88)*, pages 88-91, IEEE Computer Society Press, 1988.

SYSTEMS SOFTWARE

Our research over the past year has addressed the design of directory systems, interconnection networks, and processing elements for large-scale multiprocessors with coherent caches. Because the building-sized American Resource Computer must exploit locality to scale far beyond the limits of current multiprocessors, a major part of our effort was devoted to the issue of locality. Briefly, we have developed a model of memory referencing locality to analyze address streams of existing parallel applications, modified our Mul-T compiler and run-time system to derive statistics on locality patterns in multiprocessor applications, derived new performance evaluation models that capture the effects of locality. We are also investigating efficient performance analysis and data collection techniques for large-scale multiprocessors, and task scheduling strategies to enhance locality.

Continuing our efforts in parallel trace data collection, we now have a tracer called TMul-T, that can generate traces for parallel symbolic applications. An implementation of TMul-T on the Encore Multimax runs with a slowdown of less than 30× on a single processor. A port of TMul-T to the DEC MICROVAX is also complete. We are now porting TMul-T to the MIPS processor. We have gathered several large traces of symbolic applications written in Mul-T including MODSIM -- a functional simulator, BOYER -- a theorem prover, and several smaller applications. We have continued tracing parallel C applications under the MACH operating system using the VAX T bit technique. In a joint effort with IBM, we have derived large parallel FORTRAN traces using a postmortem scheduling method that can incorporate multiple synchronization models. FORTRAN traces include SIMPLE, WEATHER, and FFT. We are using these traces in a wide variety of studies ourselves, and we also plan to distribute our trace data to the research community and to industry. A slight modification to our parallel TMul-T tracer has also enabled the emulation of large-scale multiprocessors.

We now have running simulators for cache/directory systems and interconnection networks. These two simulators can be plugged back to back to provide the system backend to a processor emulator. Currently we can use either our TMul-T system or the FORTRAN post-mortem scheduler as the processor emulator.

We have a new model representing memory referencing locality in multiprocessor systems. This locality model suitable for multiprocessor cache evaluation is derived by viewing memory references as streams of processor identifiers directed at specific cache/memory blocks. This viewpoint differs from the traditional uniprocessor approach that uses streams of addresses to different blocks emanating from specific processors. Our view is based on the intuition that cache coherence traffic in multiprocessors is largely determined by the number of processors accessing a location, the frequency with which they access the location, and the sequence in which their accesses occur. The specific locations accessed by each processor, the time order of access to different locations, and the size of the working set play a smaller role in determining the cache coherence traffic, although they still influence intrinsic cache performance. We have some initial results that show that these processor references directed to a memory blocks display the LRU stack property. If we succeed in showing this is indeed true across a large set of parallel applications, then the abundant literature on LRU stack evaluation for single processors can be straightforwardly used in evaluation of multiprocessor performance.

We are investigating novel VLSI processor architectures for large-scale multiprocessor systems. A processor called APRIL is being designed. (This processor borrows heavily from the MARCH processor design of Bert Halstead at the MIT Laboratory for Computer Science, and the Stanford MIPS-X processor design, but differs substantially from the two. Unlike MARCH, APRIL has hardware interlocks in the pipeline, does not interleave process threads, and uses software thread scheduling. Unlike MIPS-X, it allows multiple hardware contexts, and has hardware support for synchronization and futures.) The chief issues being addressed in this design are rapid context switching, fast trap handling, high single thread performance, hardware support for synchronization and futures, and register file organization. An important observation of our study so far has been identifying the specific hardware-software tradeoffs one must make for achieving overall high system performance. Some examples include hardware versus software for fine-grain task management and scheduling in a multithreaded processor, and hardware provided synchronization primitives such as fetch-and-op versus software synthesized primitives from basic interlocked load/store instructions. We currently have a preliminary instruction-set specification. A Mul-T compiler for this processor and a simulator are also being written.

The design of a cache-directory and network communications controller to be used in a large-scale multiprocessor is in progress. The chief issues being addressed are: the programmability and the implementation efficiency of various shared-memory programming paradigms, such as strong serialization versus weak ordering; Supporting full-empty bits in the cache directory controller; Tradeoffs in controller design to support context switching, such as re-issuing instructions versus pipeline freezing.

We have analyzed interconnection network architectures that can best exploit the lower average traffic intensity of cache-coherent systems. Evaluations with packet switched and circuit switched networks, assuming similar speeds for the switch nodes, show that circuit switching can be superior to packet switching in the medium scale (256-1000 processors). Our simulations with the parallel FORTRAN traces also indicate that directories yield better processor utilization than a scheme that does not cache shared data.

We investigated the scalability of cache coherence schemes. We showed that these schemes can scale at least through 64 processors by simulations against parallel FORTRAN traces. (Memory limitations precluded our analyzing larger systems.) We observed that synchronization references are the chief impediment to scalability. We are investigating new scalable synchronization methods that do not incur excessive hardware cost.

We have developed a new technique for efficient synchronization called adaptive backoff synchronization. A purely software approach, adaptive backoff synchronization helps reduce network contention due to file-grain synchronization accesses across a network. Our technique can help reduce hot-spot contention in large-scale networks without resorting to hardware-intensive solutions like combining networks. We are also studying software combining to determine the extent to which a directory cache coherence scheme can efficiently support file-grain barrier synchronization.

Industry collaborations:

- Parallel FORTRAN applications, post-mortem scheduling,
  and address tracing with IBM T. J. Watson Research,
  Yorktown, NY. With Harold Stone, Kimming So, Scott Kirkpatrick.
- Affinity scheduling for enhancing multiprocessor memory
  referencing locality; models of multiprocessor referencing,
  software cache coherence, with DEC Systems Research Center,
  Palo Alto, CA. With Susan Owicki.
- ATUM-2 Multiprocessor data collection efforts in collaboration
  with DEC, Hudson, MA. With Dick Sites.

ALGORITHMS

Prof. Leighton is continuing his research on networks and algorithms for parallel computation. Recently he has focussed on the following specific problems: the development of fast packet routing algorithms for commonly used fixed-connection networks such as the butterfly, array and shuffle-exchange graph, the development of algorithms to reconfigure networks such as the hypercube around faults, the development of dynamic on-line algorithms for embedding computational structures such as trees in networks such as the hypercube in a way that balances computational load and that minimizes the induced communication load on the network, the development of algorithms for emulating one kind of network on another in a way that preserves the total amount of work (processors $\times$ time) that is done, and the development of efficient approximation algorithms for a variety of layout related problems such as graph bisection. The particular advances that have been made in each of these areas is briefly summarized in what follows.

In the area of packet routing, Prof. Leighton and his coauthors have discovered the first store-and-forward routing algorithm which can route n^2 packets in $2n - 2$ steps on an $n \times n$ array with constant size queues at each node. They have also discovered a more practical randomized routing algorithm that is guaranteed to have near-optimal performance for an array of any dimension (including a hypercube), the butterfly, and the shuffle-exchange graph. The latter algorithm also works for many-one routing algorithms with combining and performs well in heuristic simulations. The details of these and related results have been published.[*][†][‡]

In the area of fault-tolerance, Prof. Leighton and his coauthors have shown that a hypercube can tolerate a very large number (a constant fraction) of randomly located faults without incurring more than a constant factor loss in performance, no matter how large the hypercube is. They have also discovered simple algorithms for routing around faults in the hypercube that are guaranteed to perform nearly as well as the best routing algorithms when no faults are present. The details of this work are described elsewhere.[§]

In the area of network embeddings and scheduling, Prof Leighton and his coauthors have discovered optimal algorithms for embedding dynamically growing and shrinking trees in a hypercube so that the processing load on the nodes of the hypercube is balanced, and so that all communication links are local. This work has application to the problem of locally scheduling the work assigned to the processors of a hypercube in a dynamic fashion (i.e., as one computation spawns another, the algorithm determines the processor that will handle the new task). They have also discovered optimal algorithms for mapping code written for one architecture onto a different architecture in a way that minimizes the total amount of work required by the simulating machine. These results are described elsewhere.[||][**][††][‡‡]

---

* T. Leighton, B. Maggs and S. Rao, "Universal Packet Routing Algorithms", IEEE FOCS, pp. 256-269, October, 1988.

† T. Leighton, F. Makedon, and I. Tollis, "A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array with Constant-size Queues," ACM SPAA, to appear, June, 1989.

‡ R. Koch, "Increasing the Size of a Network by a Constant Factor Can Increase the Performance by More Than a Constant Factor," IEEE FOCS, pp. 221-230, October, 1988.

§ J. Hastad, T. Leighton, and M. Newman, "Fast Computation Using Faulty Hypercubes," ACM STOC, to appear, May, 1989.

|| S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Universal Graphs For Bounded-degree Trees and Planar Graphs," SIAM J. Discrete Math., to appear, 1989.

** R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg, "Work-preserving Emulations of Fixed-connection Networks," ACM STOC, to appear, May, 1989.

†† T. Leighton, M. Newman, and E. Schwabe, "Dynamic Embedding of Trees in Hypercubes with Constant

Lastly, in the area of approximations algorithms, Prof Leighton and Satish Rao have discovered an analogue of the max-flow min-cut theorem for multicommodity flow problems that can be used to find the first good approximation algorithms for a wide variety of NP-hard combinatorial optimization problems such as graph bisection minimum feedback arc set. This work, and some recent heuristic analysis of some related algorithms is described elsewhere.[*][†]

(continued)
Dilation and Load," ACM SPAA, to appear, June, 1989.

‡‡ S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Efficient Embedding of Trees in Hypercubes," submitted to SIAM J. Computing.

* T. Leighton and S. Rao, "An Approximate Max-flow Min-cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms," IEEE FOCS, pp. 422-431, October, 1988.

† T.Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms," DAC, to appear, June, 1989.

APPLICATIONS

In the area of applications, over the past half year our group has been working on several different types of numerical algorithms, both to aid in the design of an ARC, as well as to uncover methods that could effectively exploit an ARC. In particular, our work has focused on capacitance extraction, parallel circuit simulation, specialized circuit simulation algorithms for clocked analog circuits and analog signal processing circuits for early vision, simulation of small geometry devices, and mixed circuit/device simulation.

A fast algorithm for computing the capacitance of a complicated 3-D geometry of ideal conductors in a uniform dielectric has been developed[*]. The method is an acceleration of the standard integral equation approach for multiconductor capacitance extraction. Integral equation methods can not be applied easily to large problems because they lead to dense matrices which are typically solved with some form of Gaussian elimination. This implies the computation grows like n^3, where n is the number of tiles needed to accurately discretize the conductor surface charges. We have developed a preconditioned conjugate-gradient iterative algorithm with a multipole approximation to compute the iterates. This reduces the complexity of the multiconductor capacitance calculations to grow as $n \times m$ where m is the number of conductors.

In the area of parallelizing circuit and device simulation, the key problem is finding efficient techniques for solving large sparse linear systems. The direct or Gaussian-elimination based solution of circuit simulation matrices is particularly difficult to parallelize, mostly because the data is structured irregularly, and methods which attempt to regularize the structure, like nested dissection, lead to matrices that require much more computation to solve. We are investigating the interaction between sparse matrix data structures, computer memory structure, and multiprocessor communication (with Prof. W. Dally). One interesting recent result from simulations is that final performance is much more sensitive to communication throughput than latency.

Another approach to solving large sparse linear systems is through iteration, which is usually more structured but is not as robust as direct methods. In order to improve the convergence of relaxation methods for circuit simulation, an algorithm is being investigated that is based on extracting bands from a given sparse matrix, solving the bands directly, and relaxing on the rest of the matrix. This approach is efficient because band matrices can be solved in order $\log\ n$ time on order n processors, and this approach is more reliable than standard relaxation, because "less" relaxation is being used. A banded relaxation scheme has been developed that automatically selects the ordering of the matrix to best exploit the direct solution of the band, and to automatically select the band size[†]. In order to increase the parallelism available from the variable band algorithm, we are also investigating waveform-Newton, which allows for several timepoints of a transient simulation to be computed in parallel.

In the area of circuit simulation, the problem of simulating clocked analog circuits, like switching filters, switching power supplies, and phase-locked loops, is being attacked. These circuits are computationally expensive to simulate using conventional techniques because they are all clocked at a frequency whose period is orders of magnitude smaller than the time interval of interest to the designer. To construct such a long time solution, a program like SPICE or ASTAP must calculate the behavior of the circuit for many high frequency clock cycles. Several very efficient algorithms for theses types of problems has been developed, based on computing the solution over only a few selected high-frequency cycles. In particular, techniques for computing

---

[*] K. Nabors, J. White, "A Fast Multipole Algorithm for Capacitance Extraction of Complex 3-D Geometries,"*Proceedings, Custom Integrated Circuits Conference*, San Diego, CA, 1989.

[†] A. Lumsdaine, D. Webber, J. White, A. Sangiovanni-Vincentelli, "A Band Relaxation Algorithm for Reliable and Parallelizable Circuit Simulation," *Proceedings, International Conference on Computer-Aided Design*, Santa Clara, CA, October, 1988.

the transient behavior of switching power converters*, and computing the distortion of switched-capacitor filters have been developed†. The distortion analysis algorithm is well-suited to parallel computation as it allows many high frequency cycles to be integrated simultaneously.

A second application for specialized circuit simulation algorithms is the simulation of analog signal processing circuits used for early vision. These circuits are expensive to simulate with classical methods because they usually contain large grids of components which must be simulated at an analog level (i.e. one cannot perform simulations at a switch or gate level as is commonly done with very large digital circuits). Several properties of the analog signal processing circuits can be exploited to improve the efficiency and parallelizability of simulation algorithms. As most of these circuits are arranged in large regular grids, the computation involved is like the computations used to solve certain types of partial differential equations. We expect this research direction will lead us to generalizations of certain types of fast partial differential equation methods, and we are, in particular, focusing on waveform multigrid methods.

In the area of device simulation, we are working on simulating short-channel MOS devices. The difficulty is that the model used in conventional device simulation programs is based on the drift-diffusion model of electron transport, and this model does not accurately predict the field distribution near the drain in small geometry devices. This is of particular importance for predicting oxide breakdown due to penetration by "hot" electrons. There are two approaches for more accurately computing the electric fields in MOS devices, one is based on adding an energy equation to the drift-diffusion model and the second is based on particle or Monte-Carlo simulations.

In the first approach, an energy balance equation is solved along with the drift-diffusion equations so that the electron temperatures are computed accurately. This combined system is numerically less tame than the standard approach, and must be solved carefully. Implementations of the energy balance equation in simulators either circumvent this problem by ignoring difficult terms, or they occasionally produce oscillatory results. Research in this area is to try to develop a simulation program based on the drift-diffusion plus energy equations which is both efficient and robust. A stable numerical method for 1-D simulation has been implemented, and present work is to carry this forward to a 2-D simulator.

Work on the second approach, solving the Boltzmann equation with Monte-Carlo algorithms, is just beginning. We are focusing on issues of the numerical interaction between the computation of the self-consistent electric fields and the simulation timesteps. In addition we are investigating approaches which parallelize efficiently.

Finally, we are continuing to investigate accelerating mixed circuit/device transient simulation with waveform relaxation (WR), that is, applying WR to the sparsely-connected system of algebraic and ordinary differential equations in time generated by standard spatial discretization of the drift-diffusion equations that describe MOS devices. Recent results include proving an extension to a result indicating that the WR algorithm will converge in a uniform manner independent of the time interval, and that a multirate integration will be stable independent of timestep‡. In addition, a preliminary 2-D device simulation program based on WR has been written, and experiments on accelerating WR convergence using SOR and Conjugate-Gradient methods are in progress.

---

* K. Kundert, J. White, A. Sangiovanni-Vincentelli, "An Envelope-Following Method for the Efficient Transient Simulation of Switching Power Converters," *Proceedings, International Conference on Computer-Aided Design*, Santa Clara, CA, October, 1988.

† K. Kundert, J. White, A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Distortion Analysis of Switched Capacitor Filters," *IEEE Journal of Solid State Circuits*, April, 1989.

‡ M. Crow, J. White, M. Ilic "Stability and Convergence Aspects of Waveform Relaxation Applied to Power System Simulation," *Proceedings, International Symposium on Circuits and Systems*, Portland, OR, 1989.

Research on techniques for logic synthesis, testing and design-for-Testability focuses on the optimization of combinational and sequential circuits specified at the register-transfer or logic levels with area, performance and testability of the synthesized circuit as design parameters. The research problems being addressed are:

- Area and performance optimization of general sequential circuits composed of interacting finite state machines
- Test generation for general sequential circuits without the restrictions of Scan Design rules
- The exploration of relationships between combinational/sequential logic s[illegible]nesis and testability with a view to the development of techniques for the automatic synthesis of fully and easily testable circuits.

Interacting finite state machines (FSMs) are common in industrial chip designs. While optimization techniques for single FSMs are relatively well developed, the problem of optimization across latch boundaries has received much less attention. Techniques to optimize pipelined combinational logic so as to improve area/throughput have been proposed. However, logic *cannot* be straightforwardly migrated across latch boundaries when the basic blocks are sequential rather than combinational circuits. We have addressed the problem of logic migration across state machine boundaries so as to make particular machines less complex at the possible expense of making others more complex.[*] This can be useful from both an area and performance point of view. Optimization algorithms, based on automata-theoretic decomposition techniques, that *incrementally* modify state machine structures across latch boundaries, so as to improve area or throughput of a sequential circuit, have been developed. We are now looking toward developing more global techniques for logic migration in sequential circuits.

Interacting sequential circuits can be optimized by specifying and exploiting the don't care conditions that occur the boundaries of the different machines. While the specification of don't care conditions for interconnected combinational circuits is a well-understood problem, the corresponding sequential circuit problem has received very little attention. We have defined a *complete* set of don't cares associated with arbitrary, interconnected sequential machines. These *sequential don't cares* represent both single vectors and sequences of vectors that never occur at latch boundaries. Exploiting these don't cares can result in significant reductions in the number of states and complexities of the individual FSMs in a distributed specification. We have developed algorithms for the systematic exploitation of these don't cares[†] and are currently improving the performance of these algorithms.

Optimization of single or lumped FSMs has been the subject of a great deal of research. Optimal state assignment and FSM decomposition are critical to the synthesis of area-efficient logic circuits.

The problem of FSM decomposition entails decomposing a machine into interacting submachines so as to improve area or performance of the circuit. We have developed new decomposition techniques based on *factorization* of sequential machines.[‡],[§] This form of optimization involves identifying *subroutines* or *factors* in the original machine and *extracting* these factors to produce factored and factoring machines. Factorization can result in submachines which are smaller and faster than the original machine. Experimental results indicate

---

* S. Devadas, "Approaches to Multi-Level Sequential Logic Synthesis," *Proceedings of 26th Design Automation Conference*, Las Vegas, NV, June, 1989.

† S. Devadas, "Approaches to Multi-Level Sequential Logic Synthesis," *Proceedings of 26th Design Automation Conference*, Las Vegas, NV, June, 1989.

‡ S. Devadas and A. R. Newton, "Decomposition and Factorization of Sequential Finite State Machines," *Proceedings, International Conference on Computer-Aided Design*, Santa Clara, CA, November, 1988.

§ S. Devadas, "General Decomposition of Sequential Machines: Relationships to State Assignment," *Proceedings, 26th Design Automation Conference*, Las Vegas, NM, June, 1989.

that factorization compares favorably to other techniques for FSM decomposition. We are also currently exploring the relationships between factorization and the optimal state assignment problem.

The problem of optimal state assignment entails finding an optimal binary encoding of the states in a FSM, so the encoded and minimized FSM has minimum area. All previous automatic approaches to state encoding and assignment have involved the use of heuristic techniques. Other than the straightforward, exhaustive search procedure, no exact solution methods have been proposed. A straightforward, exhaustive search procedure requires $O(N!)$ exact Boolean minimizations, where N as the number of symbolic states. We have discovered a new minimization procedure* for multiple-valued input and multiple-valued output functions that represents an exact state assignment algorithm. The present state and next state spaces of the State Transition Graph of a FSM are treated as multiple-valued variables, taking on as many values are there are states in the machine. The minimization procedure involves constrained prime implicant generation and covering and operates on multiple-valued input, multiple-valued output functions. If a minimum set of prime implicants is selected, an minimum solution to the state assignment problem is obtained. While our covering problem is more complex than the classic unate covering problem of two-level Boolean minimization, a single logic minimization step replaces $O(N!)$ minimizations. We are currently evaluating the performance of this exact algorithm and developing computationally-efficient heuristic state assignment strategies based on the exact algorithm.

The problem of four-level Boolean minimization or the problem of finding a cascaded pair of two-level logic functions that implement another logic function, such that the sum of the product terms in the two cascaded functions or truth-tables is minimum, can also be mapped onto an encoding problem, similar to state assignment. We have extended the exact state encoding algorithm to the four-level Boolean minimization case.

After chip fabrication, a chip has to be tested for correct functionality. Logic testing is a very difficult problem and has traditionally been a post-design step; however, the impact of the design or synthesis process on the testability of the circuit is very profound.

Our research in the testing area involves test pattern generation for sequential circuits as well as the development of synthesis-for-testability approaches for combinational and sequential circuits. Highly sequential circuits, like datapaths, are not amenable to standard test pattern generation techniques. We are attempting to develop algorithms that are efficient in generating tests for datapath-like circuits, by exploiting knowledge of both the sequential behavior and the logic structure of the logic circuit.

Recently, there has been an explosion of interest in incorporating testability measures in logic synthesis techniques. Our research follows the paradigm that redundancy in a circuit, which renders a circuit untestable, is the result of a sub-optimal logic synthesis step. Thus, optimal logic synthesis can, in principle, ensure fully testable combinational or sequential logic designs.

The relationships between redundant logic and don't care conditions in combinational circuits are well known. Redundancies in a combinational circuit can be explicitly identified using test generation algorithms or implicitly eliminated by specifying don't cares for each gate in the combinational network and minimizing the gates, subject to the don't care conditions. We have explored the relationships between redundant logic and don't care conditions in arbitrary, interacting sequential circuits.†,‡ Stuck-at faults in a sequential circuit

* S. Devadas and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four-Level Boolean Minimization," Electronics Research Laboratory Memorandum M89/8, University of California, Berkeley, February, 1989.

† S. Devadas et. al., "Irredundant Sequential Machines Via Optimal Logic Synthesis," Electronics Research Laboratory Memorandum M88/52, University of California, Berkeley, August, 1988.

‡ S. Devadas et. al., "Redundancies and Don't Cares in Sequential Logic Synthesis," in preparation.

may be testable in the combinational sense, but may be redundant because they do not alter the *terminal behavior* of a non-scan sequential machine. These *sequential redundancies* result in a faulty State Transition Graph (STG) that is equivalent to the STG of the true machine. We have classified all possible kinds of redundant faults in sequential circuits, composed of single or interacting finite state machines. For each of the different classes of redundancies, we define don't care sets which if optimally exploited will result in the implicit elimination of any such redundancies in a given circuit. We have shown that the exploitation of *sequential don't cares* that correspond to sequences of vectors that never appear in cascaded or interacting sequential circuits, is critically necessary in the synthesis of irredundant circuits. Using a complete don't care set in an optimal sequential synthesis procedure of state minimization, state assignment and combinational logic optimization results in fully testable, lumped or interacting finite state machines. Preliminary experimental results indicate that irredundant sequential circuits can be synthesized with *no area overhead* and within reasonable CPU times by exploiting these don't cares.

Procedures that guarantee *easy testability* of sequential machines via constraints on the optimization steps are also a subject of research. These procedures address both the testability of circuits under the stuck-at fault and the crosspoint fault model. These procedures may result in circuits that are larger than area-minimal circuits, but which are more easily testable.

The different pieces of the research described above are all focused on an algorithmic approach for the optimal synthesis of custom integrated circuit chips with area, performance and testability as design parameters. The various techniques can be incorporated into an ASIC synthesis system.

PUBLICATIONS LIST

A. T. Ishii, "*A Digital Model for Level-Clocked Circuitry*, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, August 1988.

A. Aggarwal and J. Park, "Notes on Searching in Multidimensional Monotone Arrays," *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, October 1988.

J. Fried and B. Kuszmaul, "NAP (No ALU Processor): The Great Communicator," *Proceedings of the Second Symposium on Frontiers of Massively Parallel Computation*, October 1988.

T. Leighton, B. Maggs, and S. Rao, "Universal Packet Routing Algorithms," *Proceedings of the IEEE 29th Annual Symposium on Foundations of Computer Science*, October 1988, pp. 256- 271. Also MIT VLSI Memo No. 88-492, December 1988.

T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms," *IEEE Foundations of Computer Science*, October 1988, pp. 422-431.

T. Leighton, B. Maggs and S. Rao, "Universal Packet Routing Algorithms," *IEEE Foundations of Computer Science*, October 1988, pp. 256-269. Also, MIT VLSI Memo No. 88-492, December 1988.

R. Koch, "Increasing the Size of a Network by a Constant Factor can Increase the Performance by more than a Constant Factor," *IEEE Foundations of Computer Science*, October 1988, pp. 221-230.

S. Malitz, "Genus g Graphs have Pagenumber $O\sqrt{g}$," *IEEE Foundations of Computer Science*, October 1988, pp. 458-468.

James K. Park, "Notes on Searching in Multidimensional Monotone Arrays," *29th Annual IEEE Symposium on Foundations of Computer Science*, White Plains, New York, October 26, 1988.

Alex Ishii, "A Multi-Layer Channel Router Using One, Two, and Three Layer Partitions, or Why Four Layers of Interconnect Really Are Better Than Three," *ICCAD-88*, San Jose, California, November 8, 1988.

W. J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks," to appear in *IEEE Transactions on Computers*. Also MIT VLSI Memo No. 87-424, November 1987.

K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "An Envelope-Following Method for the Efficient Transient Simulation of Switching Power Converters," *Proceedings International Conference on Computer-Aided Design*, Santa Clara, California, November 7-10, 1988.

A. Lumsdaine, D. Webber, J. White, and A. Sangiovanni-Vincentelli, "A Band Relaxation Algorithm for Reliable and Parallelizable Circuit Simulation," *Proceedings International Conference on Computer-Aided Design*, Santa Clara, California, November 7-10, 1988.

S. Devadas and A. R. Newton, "Decomposition and Factorization of Sequential Finite State Machines," *Proceedings, International Conference on Computer-Aided Design*, Santa Clara, California, November 1988.

J. A. S. Fiske, "*A Reconfigurable Arithmetic Processor*," M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, December 16, 1988.

D. L. Standley and J. L. Wyatt, Jr., "Circuit Design Criteria for Stability in a Class of Lateral Inhibition Neural Networks," *Proceedings, IEEE Conference on Decision and Control,* Austin, Texas, December 7-9, 1988. Also MIT VLSI Memo No. 88-477, October 1988.

M. J. Foster and R. I. Greenberg,"Lower Bounds on the Area of Finite-state Machines," *Information Processing Letters*, 30(1):1--7, January 1989.

J. Fried, *VLSI Processor Design for Communication Networks*, M.S. Thesis, Department of Electrical Engineering and Computer Science, MIT, January 1989.

J. Fried, "Yield Modeling Using the SPIROS Redundancy Planner," *Proceedings of the 1st International Conference on Wafer-Scale Integration*, January 1989.

J. Park, *Notes on Searching in Multidimensional Monotone Arrays*, S.M. thesis, Department of Electrical Engineering and Computer Science, MIT, January 1989.

J. Fried, E. Daly, T. Lyszarcz, and M. Cooperman, "A Yield-Enhanced Crosspoint Switch Chip Using e-Beam Restructuring," *IEEE Transactions on Solid-State Circuits*, February 1989.

C. E. Leiserson, "VLSI Theory and Parallel Supercomputing," *Decennial Caltech VLSI Conference*, ed. C. Seitz, MIT Press, March 1989.

S. Owicki and A. Agarwal, "Evaluating the Performance of Software Cache Coherence," to appear in *ACM, Proceedings, Architectural Support for Programming Languages and Operating Systems-III*, April 1989. Also MIT VLSI Memo No. 88-478, October 1988.

K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Distortion Analysis of Switched Capacitor Filters." *IEEE Journal of Solid State Circuits*, April 1989. Also MIT VLSI Memo No. 88-480, October 1988.

R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg,"Work-preserving Emulations of Fixed-connection Networks," to appear in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, May 1989.

K. Nabors and J. White, "A Fast Multipole Algorithm for Capacitance Extraction of Complex 3-D Geometries" to appear in *Proceedings Custom International Circuits Conference*, San Diego, California, May 16-18, 1989.

M. Crow, J. White, and M. Ilic, "Stability and Convergence Aspects of Waveform Relaxation Applied to Power System Simulation," to appear in *Proceedings International Symposium on Circuits and Systems*, Portland, Oregon, May 8-11, 1989.

R. Koch, T. Leighton, B. Maggs, S. Rao and A. Rosenberg, "Work-Preserving Emulations of Fixed-Connection Networks," to appear in *ACM Symposium on Theory of Computing*, May 1989.

J. Hastad, T. Leighton, M. Newman, "Fast Computation using Faulty Hypercubes," to appear in *ACM Symposium on Theory of Computing*, May 1989..

A. Agarwal and M. Cherian, "Adaptive Backoff Synchronization Techniques," to appear in *IEEE, Proceedings of the 16th Annual Symposium on Computer Archite* *re*, , June 1989.

T. Knight, "Technologies for Low Latency Interconnection Networks," to appear in *Symposium on Parallel Algorithms and Architectures*, June 1989.

C. Phillips, "Parallel Graph Contraction," to appear in *First Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1989.

A. Agrawal, G. Blelloch, R. Krawitz, and C. Phillips, "Four Vector-matrix Primitives," to appear in *First Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1989

P. Agrawal, R. Tutundjian, and W. J. Dally, "Algorithms for Accuracy Enhancement in a Hardware Logic Simulator,"*Proceedings of the 26th ACM/IEEE Design Automation Conference*, June 1989.

T. Leighton, F. Makedon and I. Tollis, "A 2n-2 Step Algorithm for Routing in an n x n Array with Constant-Size Queues", to appear in *ACM Symposium on Parallel Algorithms and Architectures*, June 1989.

T. Leighton, M. Newman, and E. Schwabe, "Dynamic Embedding of Trees in Hypercubes with Constant Dilation and Load," to appear in *ACM Symposium on Parallel Algorithms and Architectures*, June 1989.

T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms," to appear in *Proceedings, Design Automation Conference*, June 1989.

S. Devadas, "Approaches to Multi-Level Sequential Logic Synthesis," to appear in *Proceedings of 26th Design Automation Conference*, Las Vegas, Nevada, June 1989.

S. Devadas, "General Decomposition of Sequential Machines: P.elationships to State Assignment," to appear in *Proceedings, 26th Design Automation Conference*, Las Vegas, Nevada, June 1989. Also to appear as MIT VLSI Memo No. 89-510, March 1989.

R. I. Greenberg, A. T. Ishii, and A. L. Sangiovanni-Vincentelli, "MulCh: A Multi-layer Channel Router using One, Two, and Three Layer Partitions," *IEEE International Conference on Computer-Aided Design (ICCAD-88)*, pp. 88--91, IEEE Computer Society Press, 1988.

C. Phillips and S. A. Zenios, "Experiences with Large Scale Network Optimization on the Connection Machine," *Impact of Recent Computer Advances on Operations Research*, Elsevier Science Publishing Co., New York, 1989.

W. J. Dally, "The J-Machine: System Support for Actors," *Concurrent Object Programming for Knowledge Processing: An Actor Perspective*, C. Hewitt and G. Agha editors, MIT Press, 1989. Also MIT VLSI Memo No. 88-491, December 1988. Also MIT VLSI Memo No. 88-491, December 1988.

W. J. Dally, "Mechanisms for Concurrent Computing," *Proceedings of the International Conference on Fifth Generation Computer Systems*, edited by ICOT, vol. 1, pp. 154-156, 1988.

W. Horwat, A. A. Chien, and W. J. Dally, "Experience with CST: Programming and Implementation," *Proceedings of the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, 1989.

S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Universal Graphs for Bounded-deg:ee Trees and Planar Graphs," to appear in *SIAM J. Discrete Math*, 1989.

D. L. Standley and J. L. Wyatt, Jr., "Stability Criterion for Lateral Inhibition and Related Networks that is Robust in the Presence of Integrated Circuit Parasitics," to appear in *IEEE Transactions on Circuits and Systems*, Special Issue on Neural Networks, 1989. Also MIT VLSI Memo No. 88-494, November 1988.

S. Devadas, H.-K. T. Ma, and A. R. Newton, "Easily Testable PLA-based Finite State Machines," to appear in *FTCS-19*, June 1989. Also to appear as MIT VLSI Memo No. 89-514, March 1989.

INTERNAL MEMORANDA

S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Efficient Embedding of Trees in Hypercubes," submitted to *SIAM J. Computing*, October 1988.

T. Leighton, "A 2d-1 Lower Bound for 2-Layer Knock-Knee Channel Routing," submitted to *SIAM J. Discrete Math*, November 1988.

A. Aggarwal, T. Leighton, and K. Palem, "Area-Time Optimal Circuits for Iterated Addition in VLSI," submitted to *IEEE Transactions on Computers*, November 1988.

B. Berger, M. Brady, D. Brown, and T. Leighton, "Nearly Optimal Algorithms and Bounds for Multilayer Channel Routing", submitted to *JACM*, February 1989.

A. Agarwal and A. Gupta, "Temporal, Processor, and Spatial Locality in Multiprocessor Memory References," MIT VLSI Memo No. 89-512, March 1989.

A. Aggarwal and J. Park, "Sequential Searching in Multidimensional Monotone Arrays," submitted for publication to *Algorithmica.*

A. Aggarwal and J. Park, "Parallel Searching in Multidimensional Monotone Arrays," submitted for publication to *Algorithmica.*

J. Fried and P. Kubat, "Reliability Models for Facilities Switching," submitted to *IEEE Transactions on Reliability.*

J. Kilian, S. Kipnis, and C. E. Leiserson, "The Organization of Permutation Architectures with Bussed Interconnections," MIT/LCS/TM-379, January 1989, and also VLSI-Memo 89-500, January 1989.

W. J. Dally, "Fine-Grain Concurrent Computing."

W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, M. Larivee, R. Lethin, P. Nuth, and S. Wills, "The J-Machine: A Fine-Grain Concurrent Computer."

W. J. Dally, "A Fast Translation Method for Paging on top of Segmentation."

Ronald I. Greenberg, "Area-Universal Networks," extended abstract.

TALKS WITHOUT PROCEEDINGS

Charles E. Leiserson, "Very Large Scale Computing," 25th Anniversary Symposium for Project MAC, MIT, October 1988.

J. L. Wyatt, Jr., "Design Method for Lateral Inhibition Networks that is Provably Stable in the Presence of Circuit Parasitics," October 1988, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts.

J. White,"An Envelope-Following Method for Detail Simulation of Switching Power Supplies," MIT Computers in Power Systems Conference, Cambridge, Massachusetts, October 31, 1988.

J. White, "Numerical Simulation of Switching Power and Filter Circuits," MIT Electrical Engineering Department Colloquium, November 21, 1988.

F. T. Leighton, "Packet Routing Algorithms," Distinguished Lecture, University of British Columbia, December 1988.

F. T. Leighton, "Packet Routing Algorithms for Fixed-Connection Networks," Stanford University, Palo Alto, CPA, December 1988; International Computer Science Institute, Berkeley, California, January 1989; IBM Almaden, California, January 1989.

B. Maggs, "Universal Packet Routing Algorithms," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

R. Koch, "Increasing the Size of a Network by a Constant Factor can Increase Performance by more than a Constant Factor," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

M. Cherian, "Adaptive Backoff Synchronization Techniques," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

S. Fiske, "A Reconfigurable Arithmetic Processor," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

A. Lumsdaine, "A Band Relaxation Algorithm for Reliable and Parallelizable Circuit Simulation," MIT VLSI Research Review, Cambridge, Massachusetts, December 19, 1988.

F. T. Leighton, "Networks, Parallel Computation, and VLSI Design," Distinguished Lectures, U. Oregon and NCUBE (OCATE) January 1989.

F. T. Leighton, "A Dynamic Algorithm for Embedding Trees in Hypercubes with Constant Dilation," International Computer Science Institute, Berkeley, California, January 1989.

Ronald I. Greenberg, "Area-Universal Networks," at Polytechnic University on February 6, 1989 and Princeton University on February 28, 1989.

1989 Caltech VLSI Conf.

VLSI Theory and Parallel Supercomputing

Charles E. Leiserson

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Invited Presentation

Ten years ago at this conference, Clark Thompson introduced a simple, graph-theoretic model for VLSI circuitry [22]. In Thompson's model, a circuit is a graph whose vertices correspond to active circuit elements and whose edges correspond to wires. A VLSI layout is a mapping of the graph to a two-dimensional grid, such that each vertex is mapped to a square region of the grid and each edge is mapped to a path in the grid. Unlike the classical notions of a graph embedding from mathematics, Thompson's model allows edges of a graph to cross over one another, like wires on an integrated circuit.

The interesting cost measure in VLSI is *area*. In Thompson's model, area can be measure as the number of grid points occupied by edges or vertices of the graph. Quickly, the minimum-area layouts for familiar graphs were catalogued. As shown in Figure 1, a mesh (two-dimensional array) with n vertices ($\sqrt{n}$ by $\sqrt{n}$) has $\Theta(n)$ area.[1] The normal way of drawing a complete binary tree (Figure 2a) has $\Theta(n \lg n)$ area, but the "H-tree" layout (Figure 2b) is much better: it has $\Theta(n)$ area. A hypercube, which is a popular interconnection network for parallel computers, requires considerably more area—$\Theta(n^2)$.

What causes a hypercube to occupy so much area? Although the size of a vertex grows slowly with the number of vertices in a hypercube, most of the area of a hypercube layout is devoted to *wires*. Figure 3 shows how the the problem of wiring a hypercube grows with the size of the hypercube. Wires are expensive, and wire area represents the capital cost of communication on a VLSI chip. By measuring communication costs in terms of the geometric concept of area, Thompson's model enabled a mathematical theory of communication in VLSI systems to develop.

From its origin, VLSI theory has expanded in many fruitful and interesting directions. Rather than attempting to describe the breadth of research in VLSI theory, however, I would like to revisit the accomplishments along one narrow

[1]The notation $\Theta(f(n))$ means a function that grows at the same rate as $f(n)$ to within a constant factor as n becomes large. The notation $O(f(n))$ means a function that grows no more quickly, and $\Omega(f(n))$ means a function that grows no more slowly. Formal definitions for these terms can be found in any textbook on analysis of computer algorithms.

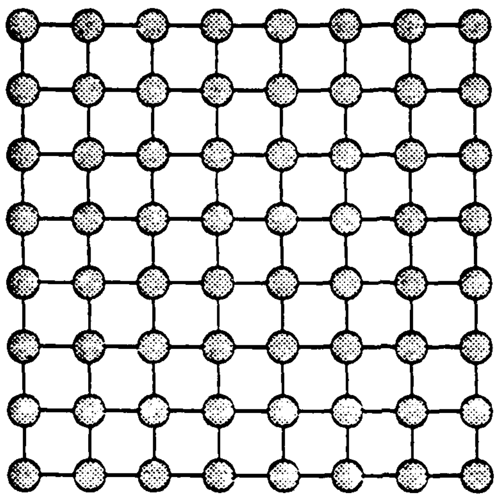

Figure 1: A mesh (two-dimensional array) on n vertices has a VLSI layout with $\Theta(n)$ area.

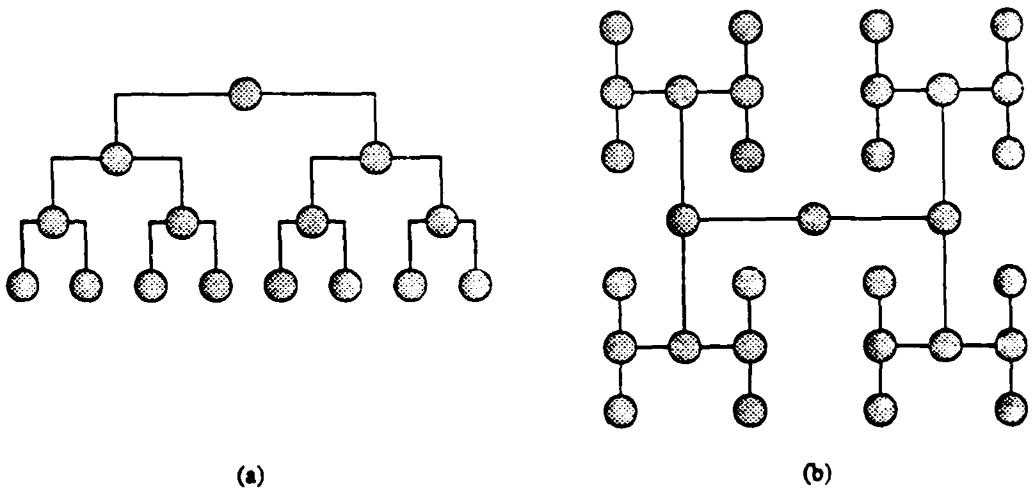

Figure 2: A complete binary tree on n vertices laid out in the standard way (a) takes $\Theta(n \lg n)$ area, but an H-tree layout (b) requires only $\Theta(n)$ area.

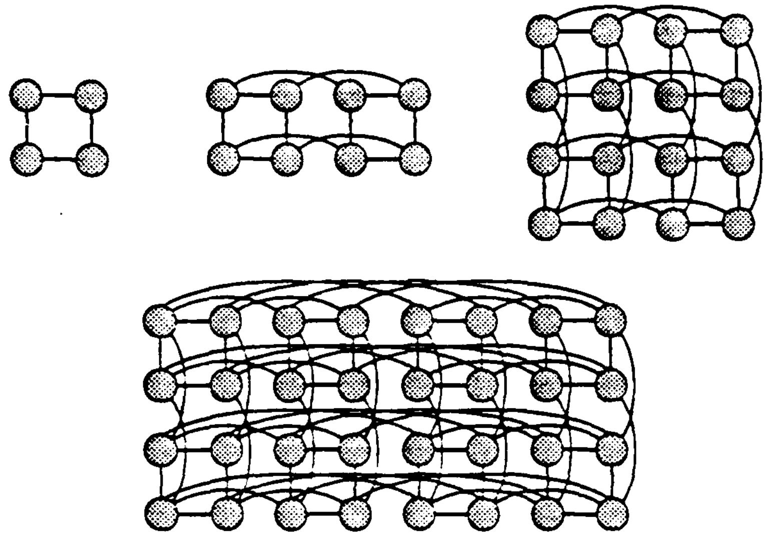
Figure 3: Illustrations (not layouts) of hypercubes on 4, 8, 16, and 32 vertices. Any layout of an n-vertex hypercube requries $\Omega(n^2)$ area.

path—layout theory—which I believe will have a fundamental impact on the architecture of large parallel supercomputers.

In his early work, Thompson discovered an important lower bound. The area of an n-vertex graph is related to its *bisection width*: the minimum number of edges that must be removed to partition the graph into two subgraphs of $n/2$ vertices (to within 1, if the number of vertices is odd). For example, an n-vertex mesh has a bisection width of $\sqrt{n}$. A complete binary tree has a bisection width of 1. A hypercube has a bisection width of $n/2$. Thompson proved that any layout of a graph with bisection width w requires $\Omega(w^2)$ area.

It turns out that a small bisection width does not lead immediately to a small-area layout. After all, if we take two $n/2$-vertex subgraphs, each with $\Theta(n^2)$ area, and connect them by a single edge, the resulting graph has a bisection width of 1 but still requires $\Theta(n^2)$ area. Leslie Valiant and I were able to show in independent work [24, 14, 15], however, that if there is a good *recursive decomposition* of a graph—one where we can keep subdividing the subgraphs without cutting many edges—then the graph has a small layout. For example, not only complete binary trees, but *any* binary tree, no matter how badly balanced, can be laid out in $O(n)$ area by a divide-and-conquer method. Valiant and I were also able to show that this method lays out any n-vertex planar graph in $O(n \lg^2 n)$ area. Later, Leighton was able to show that a variant of our method was optimal on any graph to within a $O(\lg^2 n)$ factor in area [9].

Leighton also introduced an interesting graph which he called the *tree-of-meshes* graph, shown in Figure 4. He was able to prove that this graph requires $\Omega(n \lg n)$ area, thereby refuting a conjecture of mine that all planar graphs could be laid out

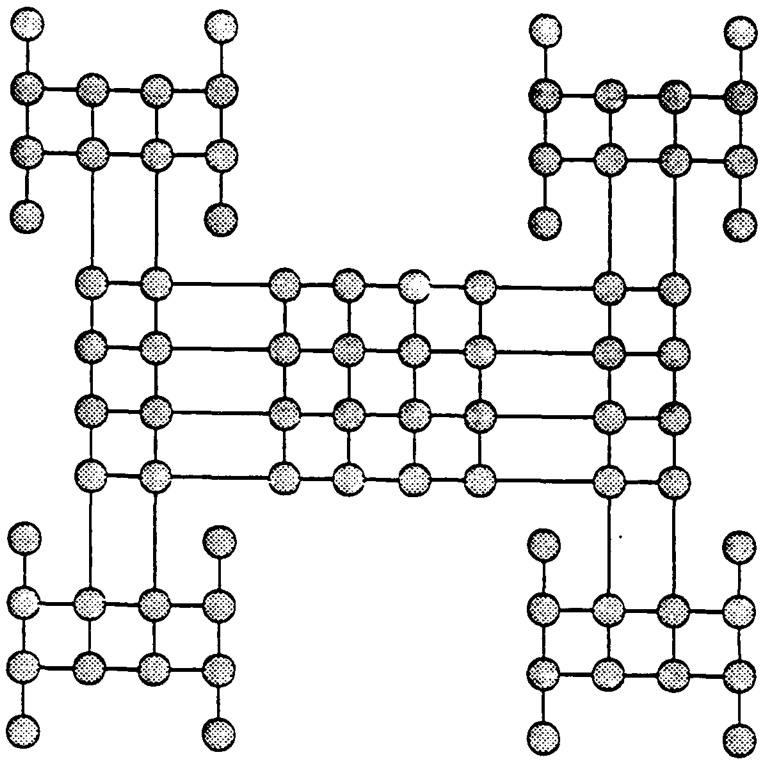

Figure 4: The tree-of-meshes graph.

in $O(n)$ area. It remains an open question in VLSI theory as to whether there exists a planar graph that requires $\Omega(n \lg^2 n)$ area, or if all planar graphs can be laid out in $O(n \lg n)$ area.

Numerous other results in layout theory have been obtained—too many to mention them all. Paterson, Ruzzo, and Snyder [18] and Bhatt and Leiserson [3] studied how to keep wires short while preserving small area. Valiant [24], Ruzzo and Snyder [21], and Dolev, Leighton, and Trickey [4] studied VLSI layouts in which wires are not allowed to cross. Three-dimensional integration was studied by Rosenberg [20], Leighton and Rosenberg [13], and Greenberg and Leiserson [5]. Fault tolerance in wafer-scale circuits was studied by Rosenberg [19], Leighton and Leiserson [11, 10], and Greene and El Gamal [7]. The packaging of graphs into chips was studied by Leiserson [15] and Bhatt and Leiserson [2].

In fact, packaging constraints are analogous to the constraints in Thompson's model. At any level of packaging—chips, boards, backplanes, racks, or cabinets—manufacturing technology constrains the number of external connections from a package to be much smaller than the number of components within the package. In Thompson's model, a square region with side s can support $4s$ external connections, but it can contain s^2 vertices, which is considerably larger than $4s$ as s becomes large.

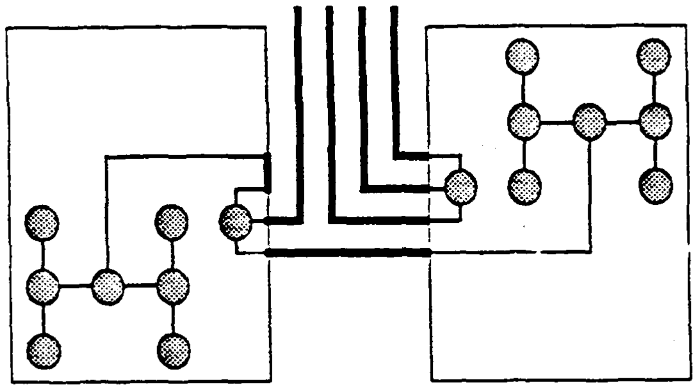

Figure 5: Packaging a complete binary tree.

As an example of a result [15] in packaging, Figure 5 shows a novel way to package a complete binary tree using 4-pin packages of a single type. Each chip contains one internal node of the tree, with three external connections, and the remainder of the chip is packed as full as possible with a complete binary tree, with one external connection. To assemble a tree with twice as many leaves, we use two chips. We wire up one of the unconnected internal nodes on one of the chips as the parent of the two complete binary trees. We are left with a complete binary tree with twice as many leaves, plus one unconnected internal node. Thus, considering the two chips as a single unit, the structure is the same as the one with which we began. By repeating the process, we can recursively assemble a complete binary tree of arbitrarily large size.

The work in layout theory culminated with the development by Bhatt and Leighton [1] of a general framework for VLSI layout. They proposed a layout method with which they were able to obtain optimal or near-optimal layouts for many graph-embedding problems. Their method has three steps. First, recursively bisect the graph, forming a decomposition tree of the graph. Second, embed the graph in the tree-of-meshes graph (Figure 4), typically, with the vertices of the graph at the leaves of the tree-of-meshes graph. The meshes in the tree-of-meshes are used as crossbar switches for routing the edges of the graph. The layout of the graph is then obtained by looking at where the vertices and edges are mapped when the the tree-of-meshes graph is laid out according to known good layouts.

It seemed to me at the time that Bhatt and Leighton had solved nearly all the interesting open problems in VLSI layout theory. All new results in the area would be little more than refinements of existing methods with no more real insights into the nature of interconnectivity. I turned my attention toward parallel computation, in which I had continued to be involved since my work with H. T. Kung on systolic arrays [8].

In fact, I was very much a proponent of special-purpose parallel computation over general-purpose parallel computation, largely as a result of my work on VLSI layout theory. After all, as Kung and I had shown, and as Kung has continued to forcefully demonstrate, many computations can be performed efficiently on simple linear-area structures such as one and two-dimensional arrays. These special-purpose networks have the nice property that they can be laid out so that processors are dense and packaging costs are minimized. Moreover, for many problems, they offer speedup which is linear in the number of processors in the systolic array.

General-purpose parallel computers, on the other hand, are typically based on interconnection networks, such as hypercubes, that are very costly for the computation they provide. For example, any hypercube network embedded in area A has at most $O(\sqrt{A})$ processors. The processors are therefore sparse in the embedding, and connections dominate the cost. Similar results can be shown for a three-dimensional VLSI model. Only $O(V^{2/3})$ processors of a hypercube network can fit in a volume V.

Hypercube networks do have a major advantage over many other networks for parallel computing, however. They are *universal*: a hypercube on n processors can simulate any n-processor bounded-degree network in $O(\lg n)$ time. The simulation overhead is polylogarithmic (a polynomial of $\lg n$), an indication that the simulation is a parallel simulation. A polynomial overhead in simulation is less interesting, since $\Theta(n)$ overhead is easily obtained by a serial processor simulating each of the n processors in turn.

The proof that an n-processor hypercube is universal goes roughly as follows. Suppose we have a bounded-degree network R with n processors. Each processor can communicate with all its neighbors in unit time. The hypercube can simulate the network, therefore, by sending at most a constant number of messages from each processor, where each message contains the information that travels on one of the interconnections in R. It turns out, all messages can be routed on the hypercube to their destinations in $O(\lg n)$ time [23].

The notion of universality—the ability of one machine to efficiently simulate every machine in a class—is central to the origins of computer science. A universal machine is the computer theorist's idea of a general-purpose, as opposed to multipurpose, machine. A universal machine can do the function of any machine, just by programming it, or, in the case of parallel-processing networks, just by routing messages. A universal machine may not be the best machine for any given job, but it is never much worse than the best. The universality theorem for hypercubes does not say that a hypercube is the fastest network to build on n processors. What it says is that the fastest special-purpose network for any given problem can't be much faster.

From a VLSI theory standpoint, however, a special-purpose parallel machine has a clear advantage over a universal parallel machine. Packaging its network can cost much less. And although universality is a selling point, our economy favors machines that are cheap and efficient, even if they are not universal. (How many combination telephone-lawnmower-toothbrushes have been sold recently?) Special-purpose networks for parallel computation are much cheaper than hypercube networks. Thus, for a long time, I was skeptical about the cost-effectiveness of general-purpose parallel computing.

I changed my mind, however, and became an advocate general-purpose parallel computing when I started to look more closely at the traditional assumptions concerning universal networks. In fact, from a VLSI theory perspective, I discovered that hypercubes are not really "universal" at all! An n-processor hypercube may be able to efficiently simulate any n-processor bounded-degree network, but if we normalize by area instead of by number of processors, we discover that an area-A hypercube cannot simulate all area-A networks efficiently. For example, since an area-A hypercube has only $\Theta(\sqrt{A})$ processors, it can't simulate an area-A mesh, which has $\Theta(A)$ processors, in polylogarithmic time. A network that is universal from a VLSI point of view should be a network that for a given area can efficiently simulate any other network of comparable area.

One such *area-universal* network is a fat-tree [16, 6], which is based on Leighton's tree-of-meshes graph. As shown in Figure 6, processors occupy the leaves of the tree. and the meshes are replaced with switches. Unlike a computer scientist's traditional notion of a tree, a fat-tree is more like a real tree in that it gets thicker further from the leaves. Local messages can be routed within subtrees, like phone calls in a telephone exchange, thereby requiring no bandwidth higher in the tree. The number of external connections from a subtree with m processors is proportional to $\sqrt{m}$, which is the perimeter of a region of area m. The area of the network is $O(n\lg^2 n)$, which is nearly linear in the number n of processors. Thus, the processors are packed densely in the layout.

Any network R that fits in a square of area n can be efficiently simulated by an area-universal fat-tree on n processors. To perform the simulation, we ignore the wires in R and map the processors of R to the processors of the fat-tree in the natural geometric way, as shown in Figure 7. As in the hypercube simulation, each wire of R is replaced by a message in the fat-tree. If we look at any m-processor subtree of the fat-tree, it simulates at most a region of area m in the layout of R. The number of wires that can leave this area-m region in R's layout is $O(\sqrt{m})$, and the fat-tree channel connecting to the root of the subtree has $\Theta(\sqrt{m})$ wires. Thus, the *load factor* of the channel, the ratio of the number of messages to channel bandwidth, is $O(1)$. It turns out that there are routing algorithms [16, 6, 12] that effectively guarantee that all messages are delivered in polylogarithmic time. (In fact, the algorithms can deliver messages near optimally even if the load factor is quite large.)

Similar universality theorems can be proved for three-dimensional VLSI models using *volume-universal* fat-trees. For a fat-tree to be universal for volume, however, the channel capacities must be selected differently from those in an area-universal network. Whereas the average growth rate of channels in an area-universal fat-tree

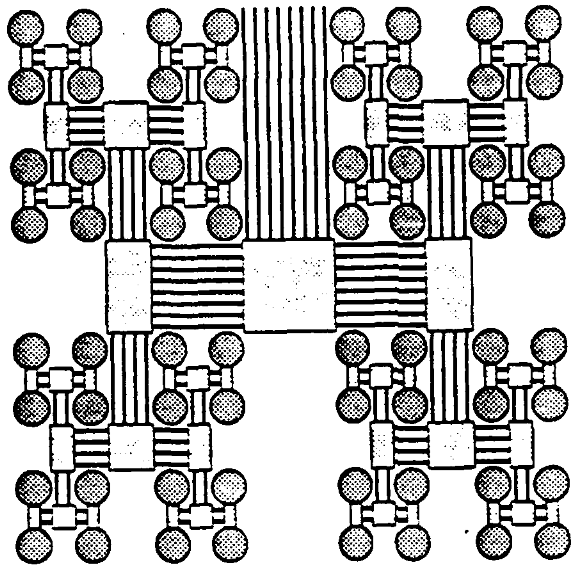

Figure 6: An area-universal fat-tree.

is $\sqrt{2}$, the average growth rate in a volume-universal fat-tree is $\sqrt[3]{4}$.

In practice, of course, no mathematical rule governs interconnect technology. Most networks that have been proposed for parallel processing, such as meshes and hypercubes, are inflexible when it comes to adapting their topologies to the arbitrary bandwidths provided by packaging technology. The growth in channel bandwidth of a fat-tree, however, is not constrained to follow a prescribed mathematical formula. The channels of a fat-tree can be adapted to effectively utilize whatever bandwidths the technology can provide and which make engineering sense in terms of cost and performance. Figure 8 shows one variant of a fat-tree composed of two kinds of small switches: a three-connection switch

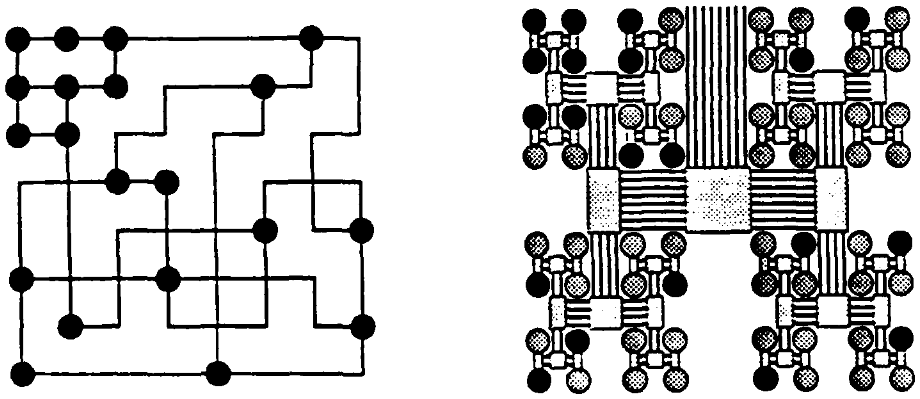

Figure 7: Any area-n network R can be efficiently simulated by an n-processor area-universal fat-tree.

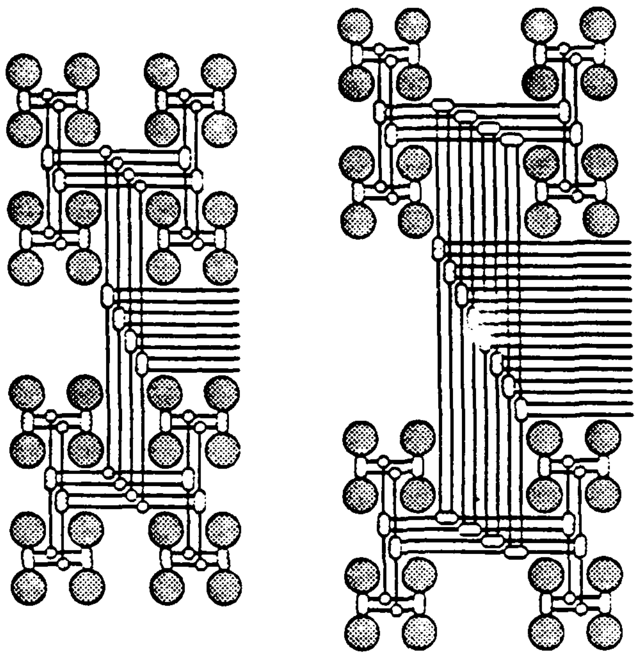

Figure 8: A scalable fat-tree.

and a four-connection switch. By choosing one of these two kinds of switches at each level of the fat-tree, the bandwidths of channels can be adjusted. If the three-connection switch is always selected, an ordinary complete binary tree results. If the four-connection switch is always selected, a butterfly network, which is a relative of a hypercube, results. By suitably mixing these two kinds of switches, a fat-tree that falls between these two extremes can be constructed that closely matches the the bandwidths provided by the interconnect technology.

The notion of locality exploited by fat-trees is but one of three such notions that arise in the engineering of a parallel computer. The most basic notion of locality is exemplified by wire delay and measured in distance. Communication is speed-of-light limited. If this notion of locality dominates, the nearest-neighbor communication provided by a three-dimensional mesh is the best one can hope. For many systems, however, wire delay is dominated by the time it takes for logic circuits to compute their functions. The second notion of locality is exemplified by levels of logic circuits and measured in gate delays. Communication time is essentially limited by the number of switches a message passes through. From this point of view, structures with small diameters, such as hypercubes, seem ideal. In a routing network, however, a heavy load of messages can cause congestion, and the time it takes to resolve this congestion can dominate both wire and gate delays. Congestion is especially likely to occur in networks that make efficient use of packaging technology. The last notion of locality is exemplified by the congestion of messages leaving a subsystem and measured by load factor. From this standpoint, fat-trees offer provably good performance by a general-purpose network that can be packaged efficiently. Recent work [17] has shown that efficient parallel algorithms can be designed for this kind of network, as well.

Whatever the point of view, however, all three notions of locality must guide the engineering and programming of very large machines. There are problems in the sciences that cry out for massive amounts of computation, most of which exhibit locality naturally: problems in astronomy, such as galaxy simulation; problems in biology, such as the combinatorics of DNA sequencing; problems in economics, such as market prediction; problems in aerospace, such as fluid-flow simulation; problems in earth, atmospheric, and ocean sciences, such as earthquake and weather prediction. To address these problems effectively, very large parallel computers must be constructed. Some of these computers may even be "building sized." To construct and program such large machines, however, locality must be exploited, and computer engineers must come to grips with the lessons of VLSI theory.

Acknowledgments

My research is supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825 and in part by an NSF Presidential Young Investigator Award with matching funds provided by IBM Corporation, AT&T Bell Laboratories, and Xerox Corporation. I am grateful to these institutions for their continuing support.

References

[1] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[2] S. N. Bhatt and C. E. Leiserson. How to assemble tree machines. *Advances in Computing Research*, 2:95–114, 1984.

[3] S. N. Bhatt and C. E. Leiserson. Minimizing the longest edge in a VLSI layout. 1981. Unpublished memorandum, MIT Laboratory for Computer Science.

[4] D. Dolev, F. T. Leighton, and H. Trickey. Planar embedding of planar graphs. *Advances in Computing Research*, 2:147–161, 1984.

[5] R. I. Greenberg and C. E. Leiserson. A compact layout for the three-dimensional tree of meshes. *Applied Mathematics Letters*, 1(2):171–176, 1988.

[6] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In *26th Annual IEEE Symposium on Foundations of Computer Science*, pages 241–249, 1985.

[7] J. W. Greene and A. El Gamal. Configuration of VLSI arrays in the presence of defects. *Journal of the ACM*, 31(4):694–717, 1984.

[8] H. T. Kung and C. E. Leiserson. Systolic arrays (for VLSI). In I. S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 256–282, SIAM, 1979.

[9] F. T. Leighton. A layout strategy for VLSI which is provably good. In *14th Annual ACM Symposium on Theory of Computing*, pages 85–98, 1982.

[10] F. T. Leighton and C. E. Leiserson. A survey of algorithms for integrating wafer-scale systolic arrays. In *IFIP Conference on Wafer-Scale Integration*, pages 177–195, 1986.

[11] F. T. Leighton and C. E. Leiserson. Wafer-scale integration of systolic arrays. *IEEE Transactions on Computers*, C-34(5):448–461, 1985.

[12] F. T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *29th Annual IEEE Symposium on Foundations of Computer Science*, pages 256–271, 1988.

[13] F. T. Leighton and A. L. Rosenberg. Three-dimensional circuit layouts. *SIAM Journal on Computing*, 15(3):793–813, 1986.

[14] C. E. Leiserson. Area-efficient graph layouts for VLSI. In *21st Annual IEEE Symposium on Foundations of Computer Science*, pages 199–214, 1980.

[15] C. E. Leiserson. *Area-Efficient VLSI Computation. ACM Doctoral Dissertation Award Series*, MIT Press, Cambridge, Massachusetts, 1983.

[16] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[17] C. E. Leiserson and B. M. Maggs. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica*, 3:53–77, 1988.

[18] M. S. Paterson, W. L. Ruzzo, and L. Snyder. Bounds on minimax edge length for complete binary trees. In *13th Annual ACM Symposium on Theory of Computing*, pages 293–299, 1981.

[19] A. L. Rosenberg. *The Diogenes approach to testable fault-tolerant networks of processors.* technical memorandum CS-1982-6.1, Department of Computer Science, Duke University, 1982.

[20] A. L. Rosenberg. Three-dimensional integrated circuitry. In H. T. Kung, R. Sproull, and G. L. Steele Jr., editors, *VLSI Systems and Computations*, pages 69–79, Computer Science Press, 1981.

[21] W. L. Ruzzo and L. Snyder. Minimum edge length planar embeddings of trees. In H. T. Kung, R. Sproull, and G. L. Steele Jr., editors, *VLSI Systems and Computations*, pages 119–123, Computer Science Press, 1981.

[22] C. D. Thompson. Area-time complexity for VLSI. In *Caltech Conference on Very Large Scale Integration*, pages 495–508, 1979.

[23] L. G. Valiant. A scheme for fast parallel computation. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[24] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C-30(2):135–140, 1981.

MECHANISMS FOR CONCURRENT COMPUTING

William J. Dally

Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

Concurrent computing is fundamentally different than sequential computing. Task size is orders of magnitude smaller making synchronization and scheduling major concerns, the critical resources are communication and memory, and programs distribute tasks rather than looping. Conventional hardware and operating system mechanisms are highly evolved for sequential computing and are not appropriate for concurrent systems. This position paper examines the mechanisms required by concurrent systems and the structure of a system incorporating these mechanisms.

1 FUNDAMENTAL PROBLEMS

1.1 Primitive Mechanisms

A fundamental hardware problem is to identify a set of primitive mechanisms that efficiently support a broad range of concurrent execution models. Sequential machines have evolved stacks for memory allocation, paging for memory management, and program counters for instruction sequencing. Concurrent machines have very different demands in each of these areas; the sequential mechanisms are no longer appropriate. However, no concurrent mechanisms have yet evolved to take their places. Today's concurrent computers either interpret their execution model using sequential mechanisms or are hardwired for a single execution model.

The message-driven processor (MDP) [4] [6] is designed to evaluate concurrent execution mechanisms for communication, synchronization, and naming. A SEND instruction and hardware message reception and buffering allow efficient communication of short messages across a high-speed network [7]. Synchronization is supported by a dispatch mechanism that creates a new process to handle a message in a single clock cycle. A general purpose translation mechanism supports naming. These mechanisms provide the primitive support required by many concurrent models of computation including dataflow [9], actors[1], and communicating processes [11].

1.2 Resource Management

At the operating system level, a key problem is to develop resource management techniques suitable for concurrent systems. In a concurrent system, communication bandwidth and memory capacity are the limiting resources; processor cycles are almost free. This situation is the opposite of the sequential case where processor cycles are considered the critical resource and communication is not a consideration. To complicate the situation the resources are physically distributed. Objects and processes must be placed in a manner that balances memory and processor use across the machine and reduces communication. The JOSS operating system [14] [15] is designed to satisfy these unconventional requirements.

Methods must also be developed to regulate concurrency. Many programs have too much parallelism and thus generate more tasks than can be accommodated in the available memory. To avoid the resulting deadlock, the system must regulate programs allowing them to generate sufficient concurrency to make use of all available processors, but reverting to more sequential execution before exhausting memory. Examples of regulation include controlled unrolling of loops [2] and adaptive (FIFO vs LIFO) scheduling [10].

To make efficient use of the communication resources, memory and tasks must be allocated in a manner that exploits locality. Placing objects near each other to improve locality is often at odds with the need to distribute objects for load balancing. Also there are some cases where communication bandwidth can be increased by spreading out a computation to make more channels available. For static computations min-cut placement techniques similar to those used to place electronic components [12] work well. Dynamic computations rely heavily on heuristics (e.g., placing an object near the object that created it) supplemented by reactive load balancing.

1.3 Overhead

To make use of a computer with thousands of processors, a program must be decomposed into many small tasks. Each task consists of only a few instructions. In conventional systems, however, the overhead of scheduling, synchronization, and communication is many hundreds of instructions per task. This overhead restricts conventional multicomputers to operating at a very coarse grain size – thousands of instructions per task. Concurrency is reduced because there are fewer large tasks. Also, the resource management problems become harder as resources are allocated in larger chunks.

Overhead can be reduced to just a few instructions per task. The JOSS operating system, using the primitive mechanisms provided by the MDP, can create, suspend, resume, or destroy a task in fewer than ten instructions [15]. This efficient management of fine-grain tasks is achieved without sacrificing protection. Each task executes in its own naming environment.

2 CONCURRENT COMPUTER ORGANIZATION

To make the most efficient use of projected VLSI technology, general purpose concurrent computers will be constructed from a number of fine-grain processing nodes [5] connected by a low-latency, wire-efficient interconnection network [3].

2.1 Fine-Grain Processing Nodes

The *grain size* of a machine refers to the physical size and the amount of memory in one processing node. A coarse-grain processing node requires hundreds of chips (several boards) and has $\approx 10^7$ bytes of memory while fine-grain node fits on a single chip and has $\approx 10^4$ bytes of memory. Fine-grain nodes cost less and have less memory than coarse-grain nodes. however, because so little silicon area is required to bu[illegible] a fast processor, they need not have slower processors than coarse-grain nodes.

VLSI technology makes it possible to build small, powerful processing elements. A 1M-bit DRAM chip has an area of 256Mλ^2 (λ is half the minimum line width [13].). In the same area we can build a single chip processing node containing:

A 32-bit processor	16Mλ^2
A floating-point unit	32Mλ^2
A communication controller	8Mλ^2
512Kbits RAM	128Mλ^2

Such a single-chip processing node would have the same processing power as a board-sized node but significantly less memory per node. The memory capacity of the entire machine is comparable to that of a coarse-grained machine. We refer to a machine built from these nodes as a *jellybean machine* as it is built with commodity part (jellybean) technology [8].

A fine-grain processing node has two major advantages: density and memory bandwidth. Several hundred single-chip nodes can be packaged on a single printed circuit board permitting us to exploit hundreds of times the concurrency of machines with board-sized nodes. With on-chip memory we can read an entire row of memory (128 or 256 bits) in a single cycle without incurring the delay of several chip crossings. This high memory bandwidth allows the memory to simultaneously buffer messages from a high bandwidth network and provide the processor with instructions and data.

Fine grain machines are area efficient. Area efficiency is given by $e_A = A_1T_1/A_NT_N$ (where A_i is the area of i processors, T_i is execution time on i processors and N is the number of processors). Many researchers have measured their machines effectiveness in terms of node efficiency, $e_N = T_1/NT_N$ Proponents of coarse-grain machines argue that a machine constructed from several thousand single-chip nodes would be inefficient because many of the processing nodes will be idle. N is large, hence e_N is small. A user, however, is not concerned with N, but rather with machine cost, A_N, and how long it takes to solve a problem, T. Fine-grain machines have a very high e_A because they are able to exploit more concurrency in a smaller area.

2.2 Wire-Efficient Communication Networks

VLSI systems are wire limited. The cost of these systems is predominantly that of connecting devices, and the performance is limited by the delay of these interconnections. Thus, an interconnection network must make efficient use of the available wire. The topology of the network must map into the three physical dimensions so that messages are not required to *double back* on themselves, and in a way that allows messages to use all of the available bandwidth along their path. Also, the topology and routing algorithm must be simple so the network switches will be sufficiently fast to avoid leaving the wires idle while making routing decisions. Our recent findings suggest that low-dimensional k-ary n-cube interconnection networks [3] are capable of providing the performance required by fine-grain concurrent architectures.

3 TRANSITION TO MAINSTREAM CONCURRENT COMPUTING

Select areas of mainstream computing will switch to concurrent computers when (1) concurrent software has matured to the point that it can support a large evolving application and (2) the performance advantage of these machines is sufficient to justify an investment in new software. Concurrent machines are appropriate for applications that are (1) limited by CPU performance (e.g., scientific computing and signal processing) and (2) limited by memory system bandwidth (e.g., transaction processing). It is also expected that the availability of these machines will create new applications that were not previously possible.

REFERENCES

References

[1] Agha, Gul A., *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, MA, 1986.

[2] Arvind, and Culler, D., "Managing Resources in a Parallel Machine", Massachusetts Institute of Technology Laboratory for Computer Science CSG Memo 257, 1985.

[3] Dally, William J. "Wire Efficient VLSI Multiprocessor Communication Networks," *Proceedings Stanford Conference on Advanced Research in VLSI,* Paul Losleben, Ed., MIT Press, Cambridge, MA, March 1987, pp. 391-415.

[4] Dally, W.J. et.al, "Architecture of a Message-Driven Processor," *Proc.* 14th *ACM/IEEE Symposium On Computer Architecture*, 1987, pp. 189-196.

[5] Dally, W.J., "Fine-Grain Concurrent Computers", *Proc.* 3rd *Symposium on Hypercube Concurrent Computers and Applications*, 1988.

[6] Dally, W.J. et.al, *Message Driven Processor Architecture, Version 11,* MIT VLSI Memo, 1988.

[7] Dally, W.J., "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, To appear.

[8] Dally, W.J., "The J-Machine", to appear.

[9] Dennis, Jack B., "Data Flow Supercomputers," *IEEE Computer*, Vol. 13, No. 11, Nov. 1980, pp. 48-56.

[10] Halstead, R., "Parallel Symbolic Computation," *IEEE Computer,* Vol. 19, No. 8, Aug. 1986, pp. 35-43.

[11] Hoare, C.A.R., "Communicating Sequential Processes," *Comm. ACM*, Vol. 21, No. 8, August 1978, pp. 666-677.

[12] Kernighan B.W. and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal,* Vol. 49, No. 2, Feb. 1970, pp. 291-307.

[13] Mead, Carver A. and Conway, Lynn A., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.

[14] Totty, B.K., *An Operating Environment for the Jellybean Machine*, MIT AI-Memo, 1988.

[15] Totty, B.K., and Dally, W.J., "JOSS: The Jellybean Operating System Software," to appear.

Experience with CST: Programming and Implementation [1]

Waldemar Horwat, Andrew A. Chien and William J. Dally
Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

CST is a programming language based on Smalltalk-80 that supports concurrency using locks, asynchronous messages, and distributed objects. In this paper, we describe CST: the language and its implementation. Example programs and initial programming experience with CST is described. An implementation of CST generates native code for the J-machine, a fine-grained concurrent computer. Some novel compiler optimizations developed in conjunction with that implementation are also described.

Introduction

This paper describes CST, an object-oriented concurrent programming language based on Smalltalk-80 [7] and an implementation of that language. CST adds three extensions to sequential Smalltalk. First, messages are asynchronous. Several messages can be sent concurrently without waiting for a reply. Second, several methods may access an object concurrently; locks are provided for concurrency control. Finally, CST allows the programmer to describe distributed objects: objects with a single name but distributed state. They can be used to construct abstractions for concurrency.

CST is being developed as part of the J-Machine project at MIT [4, 3]. The J-Machine is a fine-grain concurrent computer. The primary building block in the J-machine is the Message-Driven Processor (MDP). It efficiently executes tasks with a grain size of 10 instructions and supports a global virtual address space. This machine requires a programming system that allows programmers to concisely describe programs with method-level concurrency and that facilitates the development of abstractions for concurrency.

Object-oriented programming meets the first of these goals by introducing a discipline into message passing. Each expression implies a message send. Each message invokes a new process. Each receive is implicit. The global address space of object identifiers eliminates the need to refer to node numbers and process IDs. The programmer does not have to insert send and receive statements into the program, keep track of process IDs, and perform bookkeeping to determine which objects are local and which are remote.

For example, a CST program[2] that counts the number of leaves in a binary tree using double recursion is shown in Figure 1. Nowhere in the program does the programmer explicitly specify a send or receive, and no node numbers or process IDs are mentioned. Yet, as shown in Figure 1[3] the program exhibits a great deal of concurrency. Making message-passing implicit in the language simplifies programming and makes it easier to describe fine-grain concurrency.

CST facilitates the construction of concurrency abstractions by providing distributed objects: objects with a single name whose state is distributed across the nodes of a concurrent computer. The one-to-many naming of distributed objects along with their ability to process many messages simultaneously allows them to efficiently connect together

[1]The research described in this paper was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contracts N00014-88K-0738, N00014-87K-0825, and N00014-85-K-0124, in part by a National Science Foundation Presidential Young Investigator Award with matching funds from General Electric Corporation, an Analog Devices Fellowship, and an ONR Fellowship.

[2]This program is in *prefix* CST, a dialect that has a syntax resembling LISP. *Infix* CST [5] has a syntax closer to that of Smalltalk-80.

[3]The concurrency profiles presented in this paper are produced by an Icode level simulation of CST programs.

```
(class node (object) left right tree-node?)

(method node count-elements () ()
      (if tree-node? (+ (count-elements left)
                        (count-elements right))
          1))
```

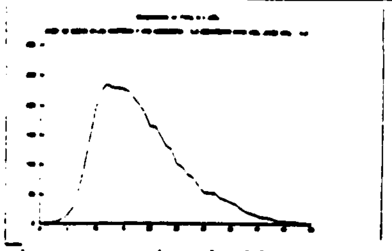

Figure 1: A CST program that calculates the number of leaves in a tree using double recursion. Its concurency profile (active tasks in each message interval) is shown to the right.

large numbers of objects. Distributing the name of a single distributed queue to sets of producer and consumer objects, for example, connects many producers to many consumers without a bottleneck.

The Optimist compiler [8] compiles Concurrent Smalltalk to the assembly language of the Message-Driven Processor (MDP) [9]. It includes many standard optimizations such as register variable assignment, dataflow analysis, copy propagation, and dead code elimination [2, 13] that are used in compilers for conventional processors. Due to the fine-grained parallel nature of the J-machine, compiling for the MDP is unlike compiling for most conventional processors in a few important aspects. For instance, loops are not important[4], while minimizing code size, tail forwarding methods, and efficiently and seamlessly handling parallelism are extremely important.

The development of Concurrent Smalltalk was motivated by dissatisfaction with process-based concurrent programming using sends and receives [11]. Many of the ideas have been borrowed from actor languages [1]. Another language named Concurrent Smalltalk has been developed at Keio University in Japan [14]. This language also allows message sending to be asynchronous, but does not include the ability to describe distributed objects.

Concurrent Smalltalk

Top-Level Expressions

A CST program consists of a number of top-level expressions. Top level forms include declarations of program and data as well as executable expressions. Linking of programs (the resolution from selectors to methods) is done dynamically.

```
<top-exp> :=    (Global <global-variable> <value>) |
                (Constant <constant-name> <value>) |
                (Class <class-name> (<superclasses>) <instance-vars>) |
                (Method <class-name> <method-name>
                        (<formals>) (<locals>)
                        <expressions>) |
                <expression>
```

Globals and Constants Globals and constant declarations define names in the environment. These names are visible in all programs, unless shadowed by a instance, argument, or local variable name. The global declaration simply defines the name. Its value remains unbound. The constant declaration defines the name and binds the name to the specified value.

Classes Objects are defined by specifying classes. Objects of a particular class have the same instance variables and understand the same messages. A class may inherit variables and methods from one or more superclasses. For

[4]In fact, the current version of Concurrent Smalltalk does not even have loops.

example:

```
(class node (object) left right tree-node?)
```

defines a class, **node**, that inherits the properties of class **object** and adds three instance variables. This means that methods for the class **node** can access all the instance variables of class **object** as well as those defined in their own class definition. Methods defined for class **object** are also inherited. Of course, this inheritance is transitive, so **node** actually inherits from a series of classes up through the top of the class hierarchy. Instance variables in the class definition may hide (shadow) those defined in the superclasses if they have the same name. The same kind of shadowing is allowed for selectors (method names).

Methods The behavior of a class of objects is defined in terms of the messages they understand. For each message, a method is executed. That execution may send additional messages, modify the object state, modify the object behavior, and create new objects. Methods consist of a header and a body. The header specifies class, selector, arguments, and locals. The body consists of one or more expressions. For example:

```
(method node count-elements () ()
      (if tree-node? (+ (count-elements left)
                        (count-elements right))
         1))
```

defines a method for class **node** with selector **count-elements**. The two empty lists indicate that there are no explicit arguments and no local variables. If present, the keyword **reply** sends the result of the following expression back to the sender of the **count-elements** message. In this case, there is no **reply** keyword, so the method replies with the value of the last expression. If the programmer wishes to suppress the reply, he can use the **(exit)** form which causes the method to terminate without a reply.

Messages are sent implicitly. Every expression conceptually involves sending a message to an object. Of course, commonly occurring special cases, like adding two local integers, will be optimized to eliminate the send. For example, **(count-elements left)**, sends the message **count-elements** to **left**. **(+ x y)** sends the message **+** with argument **x** to object **y**. If both **x** and **y** are local integers, this operation can be optimized as an add instruction.

Each expression consists of a selector, a receiver, and zero or more arguments. Identifiers must be one of: constant, global variable, argument, local variable, or instance variable. Subexpressions may be executed concurrently and are sequenced only by data dependence. For example, in the following expression from the program in Figure 1

```
(+ (count-elements left) (count-elements right))
```

the two **count-elements** messages will be sent concurrently and the **+** message will be sent when both replies have been received. The only way to serialize subexpression evaluation is to assign intermediate results to local variables.

A complete list of CST expressions is shown below:

```
<exps> := <exp>+
<exp> :=
    <name> |
    (<selector> <receiver-exp> <argument-exp>*) |
    (send <selector-exp> <receiver-exp> <argument-exp>*) |
    (value <exp>) |
    (set <name> <exp>) |
    (cset <name> <exp>) |
    (msg <node> <selector> <receiver> <actuals>) |
    (forward <continuation> <selector> <receiver> <args>) |
    (reply <exp>) |
(block (<formals>) (<locals>) <exps>) |
(if <exp> <exp> <exp>) |
(begin <exps>) |
(exit)
```

An Example CST Program

We now introduce a slightly more complicated version of the program shown in Figure 1. Rather than simply counting the leaves on a tree, we compute the lengths of all the lists linked to the tree and sums those lengths together.

```
(class node (object) left right)

(method node count-list-elements () ()
      (+ (count-list-elements left)
         (count-list-elements right)))

(class pair (object) car cdr)

(method pair count-list-elements () ()
      (length right 0 )))

(method pair length (n) ()
      (if (eq cdr 'nil) (+ 1 n)
          (length cdr (+ 1 n))))
```

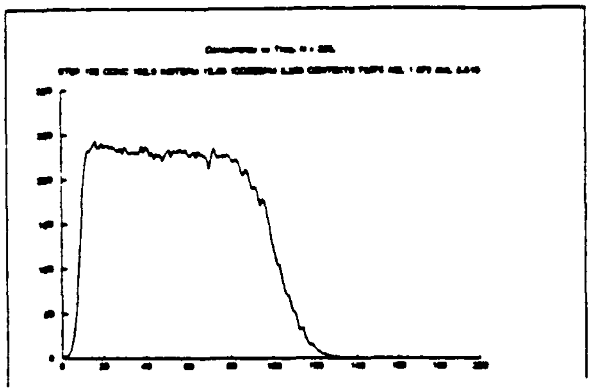

Figure 2: A CST program that computes sum of list lengths and its execution profile

The node class definition is the same as it was in Figure 1. `left` and `right` are the children of the current node in a binary tree. The right of each leaf node points to a linked list of pairs. The method `count-list-elements` recursively counts the lists lengths by doing so for the right subtree and the left subtree concurrently. At the bottom of the tree, the late binding SEND operation causes the `count-list-elements` method for pairs to be invoked. This method computes the length of each list using the tail recursive method `length`.

Distributed Objects

CST programs exhibit parallelism between objects, that is many objects may be actively processing messages simultaneously. However, ordinary objects can only receive one message at a time. CST relaxes this restriction with Distributed Objects (DOs). Distributed objects are made up of multiple representatives (constituent objects) that can each accept messages independently. The distributed object has a name (Distributed object ID or DID) and all other objects send messages to this name when they wish to use the DO.

Messages sent to the DO are received by one and only one constituent object (CO). Which constituent receives the message is left unspecified in the language. A clever implementation might send the messages to the closest constituent whereas a simpler implementation might send the messages to a random constituent. The state of a distributed object is typically distributed over the constituents. This means that responding to an external request often requires the passing of messages amongst the constituents before replying. No locking is performed on the distributed object as a whole. This means that the programmer must ensure the consistency of the distributed object.

Support for Distributed Objects

CST includes two constructs to support distributed objects. For DO creation, we add an argument for the new selector – the number of constituents desired in this DO. In order to pass messages within the object, each constituent object must be able to address each of the other constituents. This is implemented with the special selector co. Each distributed object can use this selector, the special instance variable group (a reference to the DO), and an index to address any constituent. For example, (co group 5) refers to the 5th constituent of a distributed object. Each constituent also has access to its own index and the number of constituents in the entire distributed object. Thus a description of a distributed object might look something like the example shown in Figure 3.

```
;;   Distributed Array Abstraction.
;;   The constituents are spread throughout the machine.
;;   The array state is allocated into equal sized chunks on the constituents.

(class distarray (distobj) nr-elts chunk-size elt-array)

;;
;;  Given an uninitialized DO, init makes each one an array,
;;    tells it how many elts it has, and how many elements
;;    are in the entire array.

(method distarray init (arr-size) ()
  (do-1 self (block (constit elts) ()
                    (co-init (co (group constit) (myindex constit)) elts)
                    (reply constit))
        arr-size))
;;
;; helper for init
;;
(method distarray co-init (elts) ()
        (begin (set chunk-size (/ elts (+ 1 maxindex)))
               (set nr-elts elts)
               (set elt-array (new array chunk-size))
               ))
;;
;;  Tree recursive apply, with one argument
;;
(method distarray do-1 (ablock arg1) () (1do-1 (co group 0) ablock arg1))
(method distarray 1do-1 (ablock arg1) (a b lindex rindex)
        (set lindex (lindex self))
        (set rindex (rindex self))
        (cset a (if ((<= lindex maxIndex) (1do-1 (co group lindex) ablock arg1)
                    '()))
        (cset b (if ((<= rindex maxIndex) (1do-1 (co group rindex) ablock arg1)
                    '()))
        (touch a b)
        (reply (value ablock self arg1))
        (exit))
;;
;;  Select array element at index
;;
(method distarray at (index) (selector)
        (if (or (< index (* chunk-size myindex))
                (>= index (* chunk-size (+ myindex 1))))
            (begin (set selector (truncate (/ index chunk-size)))
                   (forward requester at (co group selector) index)
                   (exit))
            (at elt-array (mod index chunk-size))))
;;
;;  Set array element at index to value
;;
(method distarray at.put (index value) (selector)
        (if (or (< index (* chunk-size myindex))
                (>= index (* chunk-size (+ myindex 1))))
            (begin (set selector (truncate (/ index chunk-size)))
                   (forward requester at.put (co group selector) index value)
                   (exit))
            (at.put elt-array (mod index chunk-size) value)))
;;
;; to make a distarray of 256 constituents and 1024 elements do
;;
;;  (init (new distarray 256) 1024)
;;
```

Figure 3: A Distributed Array Example

In the example of the distributed array, we would create a usable array with two steps. First we construct the distributed object using the new form. The example in Figure 3 creates a distributed object with 256 constituents. After the DO is created, we must initialize in a way that is appropriate for the distributed array. We do so by sending it an `init` message (also defined in Figure 3). This initialization sets each constituent up with an private array of the appropriate number of elements. For example, if we wanted a distarray of 512 elements, in this case each constituent would have a private array of two elements. This initialization is done in a tree recursive fashion and therefore takes $O(lg(n))$ time.

The mapping of the distarray elements onto the private arrays is done by the `at` and `at.put` methods. Each constituent is responsible for a contiguous range of the distarray elements. Any requests received by a constituent are first checked to see if they are within the local CO's jurisdiction. If they are not, they are forwarded to the appropriate CO. If they are, the request is handled locally. This is a particularly simple example because each constituent is wholly responsible for his subrange and need not negotiate with other constituents before modifying his local state.

Distributed objects are of great utility in building large objects on a fine grain machines. In the J-machine, we restrict ordinary objects to fit within the memory of single node, thus restricting object size. With distributed objects, we only require that a constituent of the DO fit on a single node. Some useful examples for distributed objects are dictionaries, distributed arrays, sets, queues, and priority queues.

Experience with CST

We have written a large number of Concurrent Smalltalk programs and executed them on our Icode simulator. These programs include various data structures, distributed arrays, sets, rings, B-trees, grids, and matrices. They also include several application kernels: N-body interaction and charged particle transport (Particle-in-cell algorithm). To date, the programs studied range from toys to applications of over 1000 lines. It is clear from our experience that CST programs exhibit large amounts or parallelism. However, we are just beginning to exploit the potential of Distributed Objects as building blocks for concurrent programs. We will continue to study data structures, algorithms and full-blown applications in our continuing evaluation of Concurrent Smalltalk.

The Optimist Compiler for CST

Goals

The main goal of the Optimist compiler is to produce Concurrent Smalltalk code that is as small as possible without sacrificing speed. In almost all cases optimizations that reduce space also reduce speed, but there are a few cases in which they conflict; in those cases the decisions were made in favor of optimizing space. Compilation speed was not a major goal of the compiler project; simplicity and flexibility were considered more important. Still, the compiler does achieve reasonable compilation speed, taking between one and fifteen seconds to compile most methods on a 2-megabyte Macintosh[5] II using Coral Software's Allegro Common Lisp.

Organization

The Optimist compiler is comprised of four phases, as shown in Figure 4. The Concurrent Smalltalk Front End can be replaced by other front ends to compile other languages for the MDP. Also, the Icode can be extracted from two places in the compilation process and either compiled onto different hardware or run on an Icode simulator.

The source code is converted by the Front End into an intermediate language called Icode. The Icode is at a somewhat higher level than the triples or quadruples codes that most compilers use, in that it specifies units such as entire procedure calls in single instructions. The Icode also allows for the possibility of having more than one source language compile into MDP assembly language code or having the same source language compile into several assembly languages. Figure 5 shows the `length` method in its Icode form.

[5]Macintosh is a trademark of Apple Computer, Inc.

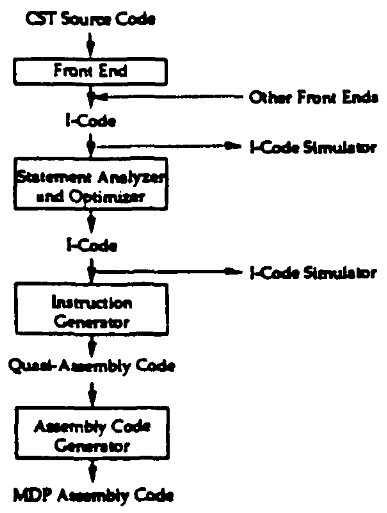

Figure 4: Compiler Organization.

```
(CSEND (TEMP 0) (METHOD EQ) (IVAR 1) (CONST NIL))
(FALSEJUMP (TEMP 0) 0)
(CSEND (TEMP 1) (METHOD +) (CONST 1) (ARG 0))
(JUMP 1)
(LABEL 0)
(CSEND (TEMP 2) (METHOD +) (CONST 1) (ARG 0))
(CSEND (TEMP 1) (METHOD LENGTH) (IVAR 1) (TEMP 2))
(LABEL 1)
(RETURN (TEMP 1))
```

Figure 5: Icode for the `length` Method: The Icode output by the Front End is a literal translation of the source code with few optimizations. At this point all method calls, including primitives, are compiled as CSENDs.

The Statement Analyzer and Optimizer processes and optimizes the Icode generated by the Front End. It performs all of the compiler's optimizations that are relevant at the Icode level of abstraction. Internally it works with Icode in the form of a directed control-flow graph. These optimizations include dead code elimination, move elimination, dataflow transformations, constant folding, tail forwarding, and merging of identical statements on both sides of paths of a conditional. The optimizations are repeatedly attempted until none of them can improve the code.

The Instruction Generator compiles each Icode statement to a number of quasi-MDP instructions and outputs the MDP code in the form of a directed control-flow graph. At the same time, the Instruction Generator assigns variables to either registers or memory locations and performs statement-specific optimizations on Icodes.

The Assembly Code Generator inserts branches into the directed graph of quasi-MDP instructions created by the Instruction Generator and performs several peep-hole optimizations. The important optimizations include shifting instructions wherever possible to align DC (Load Constant) instructions to word boundaries (all other instructions need only be aligned at half-word boundaries) and combining SEND and SENDE instructions to SEND2 and SEND2E. The Assembly Code Generator replaces short branches by long ones where necessary; such replacements are complicated by the fact that long branches alter the value of MDP's register R0. The Assembly Code Generator outputs a file of assembly language statements which can be read, assembled, and executed by our MDP simulator

MDPSim [10]. Figure 6 contains the assembly code output for the sample method **length**.

```
MODULE PAIR__LENGTH
        DC      MSG:LoadCode+18
        DC      {Class_PAIR},{Method_LENGTH}}
        MOVE    [2,A3],R0                   ;  0
        XLATE   R0,A2,XLATE_OBJ             ;  0.5
        MOVE    1,R3                        ;  1
        ADD     R3,[3,A3],R2                ;  1.5
        MOVE    [3,A2],R1                   ;  2
        BNNIL   R1,^L001                    ;  2.5
        MOVE    [4,A3],R1                   ;  3
        BNIL    R1,^L002                    ;  3.5
        DC      MSG:ReplyConst+4            ;  4
        WTAG    R1,1,R3                     ;  5
        LSH     R3,-16,R3                   ;  5.5
        SEND2   R3,R0                       ;  6
        SEND    R1                          ;  6.5
        SEND2E  [5,A3],R2                   ;  7
        BR      ^L002                       ;  7.5
L001:   MOVE    [3,A2],R0                   ;  8
        CALL    Send_Node_Nr                ;  8.5
        DC      MSG:SendConst+7             ;  9
        SEND2   R1,R0                       ; 10
        DC      {Method_LENGTH}             ; 11
        SEND    R0                          ; 12
        SEND2   [3,A2],R2                   ; 12.5
        SEND    [4,A3]                      ; 13
        SENDE   [5,A3]                      ; 13.5
L002:   SUSPEND                             ; 14
        END
```

Figure 6: Final Output of the Compiler: This is the MDP assembly code into which the **length** method compiles. If the optimizations were turned off, the code size would have been 32 words, more than twice the size of the optimized code.

Optimizations

Tail Forwarder The tail forwarder performs the message-passing equivalent of tail recursion. It is often the case that the value returned by a Concurrent Smalltalk method is the value returned by the last statement of that method, and that statement is often a method call. An example of this phenomenon is a recursive definition of the **length** function in Figure 2.

If **cdr** is not equal to **nil**, the **length** method makes a recursive call and when that call returns, it immediately returns that value as the result. There is, however, no fundamental reason why **length** should wait for the result of the recursive call to **length** only to return it to the caller; on the contrary, it would be better if the recursive **length** call returned its result to the initial caller. **length** optimized this way runs in constant space instead space proportional to the list length. The Tail Forwarder performs this optimization by looking for a CSEND statement whose value is returned by a REPLY statement immediately afterwards. Such a CSEND statement is modified to inform the callee to return its result to this method's caller instead of this method.

Fork and Join Mergers These two optimizations, if they can be applied, often produce significant savings in the output code size. They try to consolidate similar statements on both sides of forks (conditionals) and joins (places where two paths of control flow merge) in the control-flow graph.

The Join Merger looks for similar statements immediately preceding each join in the control-flow graph. Here two statements are considered to be similar if they are identical or if they are both CSENDs with identical targets and the same number of arguments; the arguments themselves need not be the same. The Join Merger moves both statements after the join; if the statements were not identical, MOVEs are generated to copy any differing arguments into temporaries before the join; the combined statement after the join will use the temporaries instead of the original arguments. These MOVEs are usually later removed by the Move Eliminator. Although more than two paths of control flow can join at the same place, the Join Merger only considers them pairwise; if more than two paths can be merged, initially two will be merged, with the other ones considered in a later pass. The Fork Merger operates analogously except that it also has to be sure not to affect the value of the condition determining which branch the program will take.

The Join Merger occasionally merges two completely different method calls which happen to have the same number of arguments, but which may even call different methods (the method selector is treated as an argument like any other), a rather unexpected optimization indeed. In each branch just before the join, the resulting object code copies the differing method arguments into the MDP's registers and stores the appropriate method selector in a register. After the join is common code that sent the message given the method selector and arguments in the registers. Since the code to send a message is long compared to the code to load values into registers, the optimization has a net savings of five words (ten instructions) of code without significantly affecting the running time.

Move Eliminator For each MOVE statement from a local variable to another local variable, the Move Eliminator attempts to merge the source and destination variables into one variable and then remove the MOVE statement. Such a merge can be done successfully if the two variables are never simultaneously live at any point in the code.

The Move Eliminator complements the copy-propagation algorithm in the Optimist. Although both try to optimize MOVE statements, each is able to handle cases that the other cannot. The copy propagation can handle constants, while Figure 7 shows an example of MOVE statements that can be eliminated by the Move Eliminator but not by copy propagation.

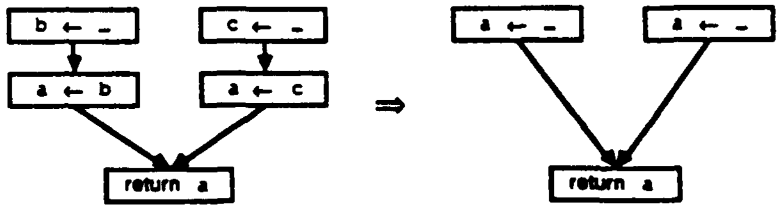

Figure 7: Move Eliminator Example: The Move Eliminator is able to remove the two MOVE statements (a←b) and (a←c) in the above code (the arrows indicate possible flow of control paths). The copy propagation algorithm would not detect the opportunity to remove these two MOVE statements because the value of a at the return statement is neither a copy of b nor a copy of c. The above code does occur in many methods.

Variable Allocator A greedy algorithm is used to assign eligible variables to registers. The shortest-lived variables with the most references are considered first. A graph coloring algorithm is used to assign the variables that did not fit in the registers to context slots; thus, fewer context slots are used, saving valuable memory space.

Summary

In this paper, we have presented a new language, Concurrent Smalltalk, that is designed for concurrency. Specific support for concurrency includes locks, distributed objects, and asynchonous message passing.

Distributed Objects represent a significant innovation in programming parallel machines. We refer to the constituents of a distributed object with a single name, but the implementation of the object is with many constituents. This different perspective allows easy use of distributed objects by outside programs while allowing the exploitation of internal concurrency.

We have described an implementation of a CST system. This programming environment includes a compiler, simulator, and statistics collection package. This set of tools allows us to experiment with new constructs and implementation techniques for the language. Although many of the optimizations used by the Optimist compiler are generally known, they have usually been applied to compilers for conventional processors. The issues involved in compiling for the MDP are quite different from compiling for conventional processors. After examining the compiler's output, it becomes apparent that the optimizations are essential to the successful use of Concurrent Smalltalk on the MDP. The compiler's optimizations reduce the amount of code output by anywhere between 20% and 60% (or even more in some cases) compared to output with all nonessential optimizations disabled. Such a reduction is very important on a processor with only 4096 words of primary memory.

There are many open issues relating to CST and similar programming systems. Key efficiency issues remain unresolved: how fine grain will the programs written in CST be and what is the run time overhead of CST programs? There are also concerns about the expressive power of languages like CST – how easy is it to write programs in CST and how useful are distributed objects?

References

[1] Agha, Gul A., *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, MA, 1986.

[2] Aho, Alfred V., Sethi, Ravi, and Ullman, Jeffrey D. *Compilers: Principles, Techniques, and Tools* Addison-Wesley, Reading, MA, 1986.

[3] Dally, W. J., "Fine-Grain Message-Passing Concurrent Computers," *these proceedings.*

[4] Dally, William J. et.al., "Architecture of a Message-Driven Processor," *Proceedings of the 14th ACM/IEEE Symposium on Computer Architecture*, June 1987, pp. 189-196.

[5] Dally, William J., *A VLSI Architecture for Concurrent Data Structures*, Kluwer, Boston, MA, 1987.

[6] Halstead, Robert H., "Parallel Symbolic Computation," *IEEE Computer*, Vol. 19, No. 8, Aug. 1986, pp. 35-43.

[7] Goldberg, Adele and Robson, David, *Smalltalk-80, The Language and its Implementation*, Addison Wesley, Reading MA, 1984.

[8] Horwat, Waldemar, *A Concurrent Smalltalk Compiler for the Message-Driven Processor*, MIT AI Technical Report 1080, October 1988.

[9] Horwat, Waldemar and Totty, Brian. *Message-Driven Processor Architecture, Version 10* MIT Concurrent VLSI Architecture Memo, March 1988.

[10] Horwat, Waldemar and Totty, Brian. *Message-Driven Processor Simulator, Version 5.0* MIT Concurrent VLSI Architecture Memo, December 1987.

[11] Su, Wen-King, Faucette, Reese, and Seitz, Charles L., *C Programmer's Guide to the Cosmic Cube*, Technical Report 5203:TR:85, Dept. of Computer Science, California Institute of Technology, September 1985.

[12] Totty, Brian, "An Operating System Kernel for the Jellybean Machine," *MIT Concurrent VLSI Architecture Memo*, 1987.

[13] Wulf, William M., Johnsson, Richard K., Weinstock, Charles B., Hobbs, Steven O., and Geschke, Charles, M. *The Design of an Optimizing Compiler*. American Elsevier, New York, 1975.

[14] Yokote, Yasuhiko and Tokoro, Mario, "Concurrent Programming in ConcurrentSmalltalk," *Object-Oriented Concurrent Programming*, Yonezawa and Tokoro eds., MIT Press, Cambridge, MA, 1987, pp. 129-158.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-477

October 1988

CIRCUIT DESIGN CRITERIA FOR STABILITY IN A CLASS OF LATERAL INHIBITION NEURAL NETWORKS

D. Standley and J. L. Wyatt, Jr.

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values of parasitic elements may be unknown. The method is based on a mathematical analysis using Tellegen's theorem and the Popov criterion. The criteria are *local* in the sense that no overall analysis of the interconnected system is required for their use, *empirical* in the sense that they involve only measurable frequency response data on the individual cells, and *robust* in the sense that they are not affected by unmodelled parasitic resistances and capacitances in the interconnect network.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

Acknowledgements

To appear in Proceedings, IEEE Conference on Decision and Control; Austin, TX, December 7-9, 1988. This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825, and by the National Science Foundation under contract number MIP-8814612.

Author Information

Standley: Department of Electrical Engineering and Computer Science, Room 36-863, MIT, Cambridge, MA 02139, (617) 253-2631.
Wyatt, Jr.: Department of Electrical Engineering and Computer Science, Room 36-864, MIT, Cambridge, MA 02139, (617) 253-6718.

To appear in Proc. IEEE Conference on Decision and Control; Austin, TX, December 7-9, 1988.

Circuit Design Criteria For Stability In A Class Of Lateral Inhibition Neural Networks

D. Standley and J.L. Wyatt, Jr.

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values of parasitic elements may be unknown. The method is based on a mathematical analysis using Tellegen's theorem and the Popov criterion. The criteria are *local* in the sense that no overall analysis of the interconnected system is required for their use, *empirical* in the sense that they involve only measurable frequency response data on the individual cells, and *robust* in the sense that they are not affected by unmodelled parasitic resistances and capacitances in the interconnect network.

I. Introduction

The term "lateral inhibition" first arose in neurophysiology to describe a common form of neural circuitry in which the output of each neuron in some population is used to inhibit the response of each of its neighbors. Perhaps the best understood example is the horizontal cell layer in the vertebrate retina, in which lateral inhibition simultaneously enhances intensity edges and acts as an automatic gain control to extend the dynamic range of the retina as a whole [1]. The principle has been used in the design of artificial neural system algorithms by Kohonen [2] and others and in the electronic design of neural chips by Carver Mead et. al. [3,4].

In the VLSI implementation of neural systems, it is convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. Linear resistors fabricated in, e.g., polysilicon, could yield a very compact realization, and nonlinear resistive grids, made from MOS transistors, have been found useful for image segmentation [4,5]. Networks of this type can be divided into two classes: feedback systems and feedforward-only systems. In the feedforward case one set of amplifiers imposes signal voltages or currents on the grid and another set reads out the resulting response for subsequent processing, while the same amplifiers both "write to" the grid and "read from" it in a feedback arrangement. Feedforward networks of this type are inherently stable, but feedback networks need not be.

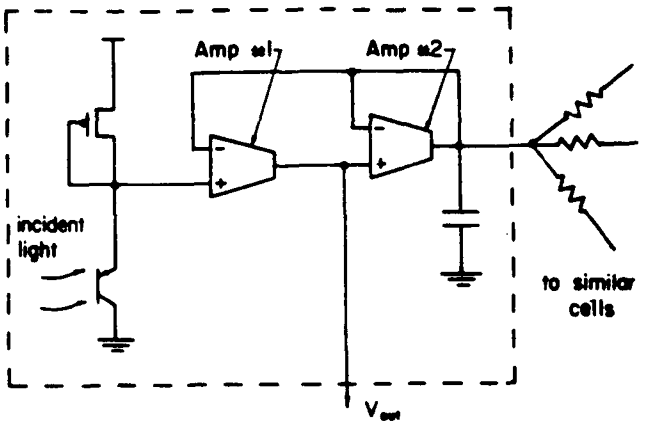

Figure 1: This photoreceptor and signal processor circuit, using two MOS amplifiers, realizes lateral inhibition by communicating with similar cells through a resistive grid.

A practical example is one of Carver Mead's retina chips [3] that achieves edge enhancement by means of lateral inhibition through a resistive grid. Figure 1 shows a single cell in a continuous-time version of this chip, and Fig. 2 illustrates the network of interconnected cells. Note that the voltage on the capacitor in any given cell is affected both by the local light intensity incident on that cell and by the capacitor voltages on neighboring cells of identical design. Each cell drives its neighbors, which drive both their distant neighbors and the original cell in turn. Thus the necessary ingredients for instability — active elements and signal feedback — are both

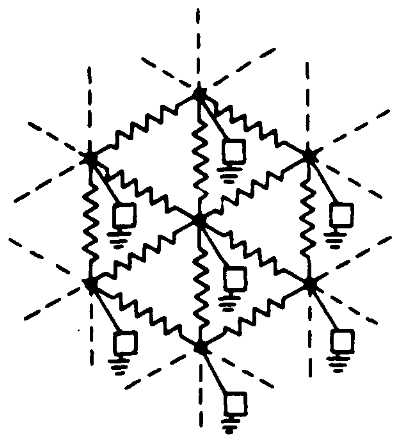

Figure 2: Interconnection of cells through a hexagonal resistive grid. Cells are drawn as 2-terminal elements with the power supply and signal output lines suppressed. The grid resistors will be nonlinear by design in many such circuits.

present in this system. Experiment has shown that the individual cells in this system are open-circuit stable and remain stable when the output of amp # 2 is connected to a voltage source through a resistor, but the interconnected system oscillates so badly that the original design is essentially unusable in practice with the lateral inhibition paths enabled [6]. Such oscillations can readily occur in most resistive grid circuits with active elements and feedback, even when each individual cell is quite stable. Analysis of the conditions of instability by conventional methods appears hopeless, since the number of simultaneously active feedback loops is enormous.

This paper reports a practical design approach that rigorously guarantees such a system will be stable. The work begins with the naivé observation that the system would be stable if we could design each individual cell so that, although internally active, it acts like a passive system as seen from the resistive grid. The design goal in that case would be that each cell's output impedance should be a *positive-real* [7,8, and 9, p. 174] function. This is sometimes possible in practice; we will show that the original network in Fig. 1 satisfies this condition in the absence of certain parasitic elements. Furthermore, it is a condition one can verify experimentally by frequency-response measurements.

It is obvious that a collection of cells that appear passive at their terminals will form a stable system when interconnected through a passive medium such as a resistive grid, and that the stability of such a system is *robust* to perturbations by passive parasitic elements in the network. The contribution of this paper is to go beyond that observation to provide i) a demonstration that the passivity or positive-real condition is much stronger than we actually need and that weaker conditions, more easily achieved in practice, suffice to guarantee robust stability of the linear network model, and ii) an extension of the analysis to the *nonlinear* domain that furthermore rules out sustained *large-signal* oscillations under certain conditions.

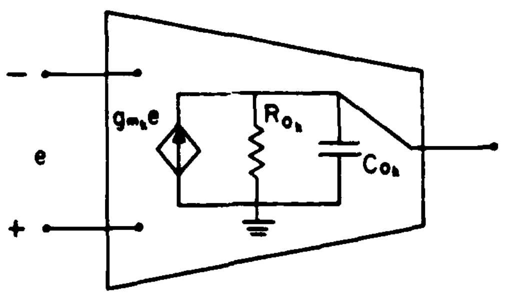

Figure 3: Elementary model for an MOS amplifier. These amplifiers have a relatively high output resistance, which is determined by a bias setting (not shown).

Note that the work reported here does not apply directly to networks created by interconnecting neuron-like elements, as conventionally described in the literature on artificial neural systems, through a resistive grid. The "neurons" in, e.g., a Hopfield network [10] are *unilateral 2-port elements* in which the input and output are both voltage signals. The input voltage uniquely and instantaneously determines the output voltage of such a neuron model, but the output can only affect the input via the resistive grid. In contrast, the cells in our system are *1-port electrical elements* (temporarily ignoring the optical input channel) in which the port voltage and port current are the two relevant signals, and each signal affects the other through the cell's internal dynamics (modelled as a Thevénin equivalent impedance) as well as through the grid's response.

II. The Linear Theory

This work was motivated by the following linear analysis of a model for the circuit in Fig. 1. For an initial approximation to the output admittance of the cell we use the elementary model shown in Fig. 3 for the amplifiers and simplify the circuit topology within a single cell (without loss of relevant information) as shown in Fig. 4.

Straightforward calculations show that the output admittance is

$$Y(s) = [g_{m_2} + R_{o_2}^{-1} + sC_{o_2}] + \frac{g_{m_1} g_{m_2} R_{o_1}}{(1 + sR_{o_1} C_{o_1})}\ , \qquad (1)$$

which is positive-real.

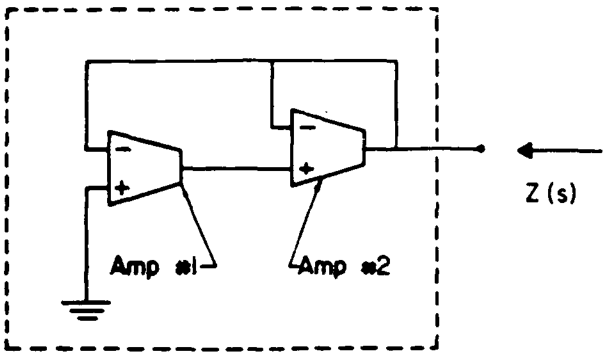

Figure 4: Simplified network topology for the circuit in Fig. 1. The capacitor that appears explicitly in Fig. 1 has been absorbed into C_{o_2}.

Of course this model is oversimplified, since the circuit *does* oscillate. Transistor parasitics and layout parasitics cause the output admittance of the individual active cells to deviate from the form given in eq. (1), and any very accurate model will necessarily be quite high order. The following theorem shows how far one can relax the positive-real condition and still guarantee that the entire network is robustly stable.

Terminology

The terms *open right-half plane* and *closed right-half plane* refer to the set of all complex numbers $s = \sigma + j\omega$ with $\sigma > 0$ and $\sigma \geq 0$, respectively, and the term *closed second quadrant* refers to the set of complex numbers with $\sigma \leq 0$ and $\omega \geq 0$. A *natural frequency* of a linear network is a complex frequency s_o such that, when all independent sources are set to zero and all branch impedances and admittances are evaluated at s_o, there exists a nonzero solution for the complex branch voltages $\{V_k\}$ and currents $\{I_k\}$ [11]. A lumped linear network is said to be *stable* if a) it has no natural frequencies in the closed right-half plane except perhaps at the origin, and b) any natural frequency at the origin results only in network solutions that are constant as functions of time. (The latter condition rules out unstable transient solutions that grow polynomially in time resulting from a repeated natural frequency at the origin.)

Theorem 1

Consider the class of linear networks of arbitrary topology, consisting of any number of positive 2-terminal resistors and capacitors and of N lumped linear impedances $Z_n(s), n = 1,2,...,N$, that are open- and short-circuit stable in isolation, i.e., that have no poles or zeroes in the closed right-half plane. Every such network is *stable* if at each frequency $\omega \geq 0$ there exists a phase angle $\theta(\omega)$ such that $0 \geq \theta(\omega) \geq -90°$ and $|\angle Z_n(j\omega) - \theta(j\omega)| < 90°, n = 1,2,...,N$.

An equivalent statement of this last condition is that the Nyquist plot of each cell's output impedance for $\omega \geq 0$ never intersects the closed 2nd quadrant, and that no two cells' output impedance phase angles can ever differ by as much as 180°. If all the active cells are designed identically and fabricated on the same chip, their phase angles should track fairly closely in practice, and thus this second condition is a natural one.

The theorem is intuitively reasonable. The assumptions guarantee that the cells cannot resonate with one another at any purely sinusoidal frequency $s = j\omega$ since their phase angles can never differ by as much as 180°, and they can never resonate with the resistors and capacitors since there is no $\omega \geq 0$ at which both $Re\{Z_n(j\omega)\} \leq 0$ and $Im\{Z_n(j\omega)\} \geq 0$ for some $n, 1 \leq n \leq N$. The proof formalizes this argument using conservation of complex power, extends it to rule out natural frequencies in the right-half plane as well, and shows why instabilities resulting from a repeated natural frequency at the origin cannot occur.

Proof of Theorem 1

Let s_o denote a natural frequency of the network and $\{V_k\}, \{I_k\}$ denote any complex network solution at s_o. By Tellegen's theorem [12], or conservation of complex power, we have

$$\sum_k V_k I_k^* = 0, \tag{2}$$

i.e., for $s_o \neq$ any pole of $Z_n, n = 1,...,N$ and $s_o \neq 0$,

$$\underset{\text{resistors}}{\sum} |I_k|^2 R_k + \underset{\text{capacitors}}{\sum} |I_k|^2 (s_o C_k)^{-1} + \underset{\text{cells}}{\sum} |I_n|^2 Z_n(s_o) = 0 \tag{3}$$

and for $s_o \neq$ any zero of $Z_n, n = 1,...,N$,

$$\underset{\text{resistors}}{\sum} |V_k|^2 R_k^{-1} + \underset{\text{capacitors}}{\sum} |V_k|^2 s_o^* C_k + \underset{\text{cells}}{\sum} |V_k|^2 Y_n^*(s_o) = 0 \ , \tag{4}$$

where the superscript $*$ denotes the complex conjugate operation. The proof is completed in the following three parts, which together rule out the existence of any natural frequencies in the closed right-half plane (except possibly for a single one at the origin).

Part i)

This part shows that there are no natural frequencies at $s_o = j\omega \neq 0$. For each $\omega > 0$ all the cell impedance values

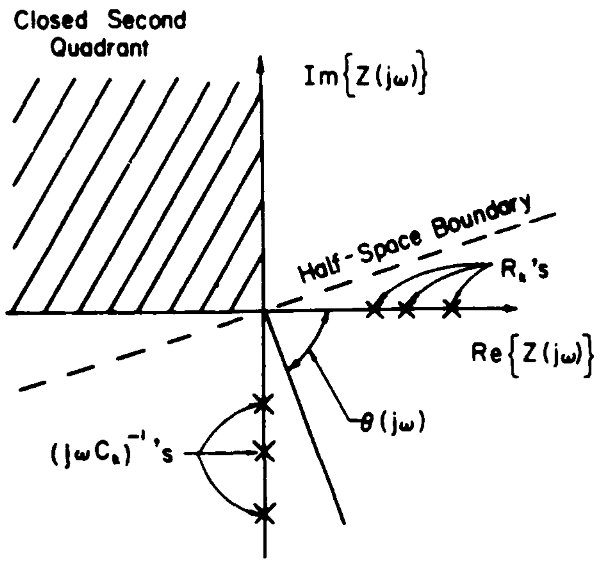

Figure 5: Illustration for the proof of Theorem 1.

lie strictly below and to the right of a half-space boundary passing through the origin of the complex plane at an angle $\theta(\omega) + 90°$ with the real positive axis, as shown in Fig. 5. The capacitor impedances $\{(j\omega C_k)^{-1}\}$ and the resistor impedances $\{R_k\}$ also lie below and to the right of this line. Thus no positive linear combination of these impedances can vanish as required by (3). A similar argument holds for $\omega < 0$.

Part ii)

This part shows that there cannot exist a repeated natural frequency at the origin that leads to a time-dependent solution. The assumptions that the cell impedances have no $j\omega$-axis zeroes and that their Nyquist plots for $\omega \geq 0$ never intersect the closed 2nd quadrant imply that $Y_n^*(0) > 0$, $n = 1, ..., N$. Thus (4) requires that all the voltages across resistor branches and cell output branches must vanish in any complex network solution at $s_o = 0$. Thus only capacitor voltages can be nonzero and the network solution will be unaltered if all non-capacitor branches are replaced by short circuits. But every solution to a network comprised only of positive, linear 2-terminal capacitors is constant in time (and hence stable).

Part iii)

This part uses a homotopy argument to show that there are no natural frequencies in the open right-half plane. Assume the contrary, i.e., that there exists such a network with a natural frequency s_1 with $Re\{s_1\} > 0$. Alter each element in the network (except resistors) as follows. For each cell having a $Z_n(s)$ of relative degree less than zero, add a series resistance R; for all other cells and for capacitors, add a parallel conductance G to each. Call each resulting pair a "composite element", and choose $R = G = \lambda \geq 0$. For λ sufficiently large all natural frequencies must lie in the open left-half plane since every branch element is strictly passive for λ sufficiently large. Since the natural frequencies are continuous functions of λ [13] and $Re\{s_1\} > 0$ for $\lambda = 0$, there exists some $\acute{\lambda} > 0$ for which some natural frequency $\acute{s}_1$ lies on the imaginary axis. But this is ruled out by the proof in part i) unless $\acute{s}_1 = 0$, and the argument in part ii) rules out $\acute{s}_1 = 0$, since any network solution at $\acute{s}_1 = 0$ consists of zero branch voltages except for capacitor branches, and for $\lambda > 0$ each capacitor has a positive conductance G in parallel with it. Since the voltage across every G is zero in such a network solution, all branch voltages (and thus all branch currents) in that solution must be zero, which is a contradiction because a natural frequency at $\acute{s}_1$ implies the existence of a nonzero solution. ■

III. Stability Result for Networks with Nonlinear Resistors and Capacitors

The previous results for linear networks can afford some limited insight into the behavior of nonlinear networks. If a linearized model is stable, then the equilibrium point of the original nonlinear network is *locally stable*. But the result in this section, in contrast, applies to the full nonlinear circuit model and allows one to conclude that in certain circumstances the network cannot oscillate *even if* the initial state is *arbitrarily far from* the equilibrium point.

Terminology

We say that a function $y = f(x)$ *lies in the sector* [a,b] if $ax^2 \leq xf(x) \leq bx^2$. And we say that an impedance $Z(s)$ satisfies the *Popov criterion* if $(1 + \tau s)Z(s)$ is *positive real* [7,8,and 9, p. 174] for some $\tau > 0$. (Note that this formulation of the Popov criterion differs slightly from that given in standard references [8 and 9, p. 186].)

Theorem 2

Consider a network consisting of possibly nonlinear resistors and capacitors and cells with linear output impedances $Z_n(s), n = 1, 2, ..., N$. Suppose

i) the resistor curves are continuous functions $i_k = g_k(v_k)$ where g_k lies in the sector $[0, G_{max}], G_{max} > 0$, for all resistors,

ii) the capacitors are characterized by continuous functions $i_k = C_k(v_k)\dot{v}_k$ where $0 \leq C_k(v_k) \leq C_{max}$ for all k and v_k, and

iii) the impedances $Z_n(s)$ all satisfy the Popov criterion for some common value of $\tau > 0$. Then the network is stable in the sense that, for any initial condition,

$$\int_0^\infty \Big[\sum_{\substack{\text{all resistors} \\ \text{and capacitors}}} i_k^2(t) \Big] dt < \infty . \qquad (5)$$

Outline of Proof

By Tellegen's theorem, for any set of initial conditions and any time $T > 0$,

$$\int_0^T \sum_{\text{resistors}} (v_k(t) + \tau \dot{v}_k(t)) i_k(t) dt \ +$$

$$\int_0^T \sum_{\text{capacitors}} (v_k(t) + \tau \dot{v}_k(t)) i_k(t) dt \ +$$

$$\int_0^T \sum_{\text{cell impedances}} (v_k(t) + \tau \dot{v}_k(t)) i_k(t) dt = 0. \qquad (6)$$

For resistors, multiplying the sector inequality $vg(v) \leq G_{\max} v^2$ by $\frac{i}{v} \geq 0$ yields $i^2 = ig(v) \leq G_{\max} iv$, and hence

$$G_{\max}^{-1} \int_0^T i_k^2(t) dt \leq \int_0^T i_k(t) v_k(t) dt =$$

$$\int_0^T i_k(t)[v_k(t) + \tau \dot{v}_k(t)] dt - \tau[\phi_k(v_k(T)) - \phi_k(v_k(0)] \quad (7)$$

where

$$\phi_k(v) = \int_0^v g_k(v') dv' \geq 0 \qquad (8)$$

is the *resistor co-content*. Using the inequality (8) in (7) yields for each resistor

$$G_{\max}^{-1} \int_0^T i_k^2(t) dt - \tau \phi_k(v_k(0)) \leq \int_0^T i_k(t)[v_k(t) + \tau \dot{v}_k(t)] dt. \qquad (9)$$

For capacitors, integrating the inequality $i_k^2 = C_k^2(v_k)\dot{v}_k^2 \leq C_{\max} C_k(v_k) \dot{v}_k^2$ yields

$$\frac{\tau}{C_{\max}} \int_0^T i_k^2(t) dt \leq \tau \int_0^T C_k(v_k) \dot{v}_k^2(t) dt =$$

$$\int_0^T i_k(t)[v_k(t) + \tau \dot{v}_k(t)] dt - [E_k(q_k(T) - E_k(q_k(0)], \quad (10)$$

where

$$E_k(q) = \int_0^q v_k(q') dq' \geq 0 \qquad (11)$$

is the capacitor energy. Using the inequality (11) in (10) yields for each capacitor

$$\frac{\tau}{C_{\max}} \int_0^T i_k^2(t) dt - E_k(q_k(0) \leq \int_0^T i_k(t)[v_k(t) + \tau \dot{v}_k(t)] dt. \qquad (12)$$

And for the cells, the assumption that $(1 + \tau s) Z_n(s)$ is positive real implies that

$$\int_0^T i_n(t)[v_n(t) + \tau \dot{v}_n(t)] dt \geq -E_n(0), \qquad (13)$$

where $E_n(0)$ is the initial "energy" in the mathematically constructed impedance $(1 + \tau s) Z_n(s)$ at $t = 0$, a function of the initial conditions only. Substituting (9), (12) and (13) into (6) yields

$$G_{\max}^{-1} \int_0^T \sum_{\text{resistors}} i_k^2(t) dt + \frac{\tau}{C_{\max}} \int_0^T \sum_{\text{capacitors}} i_k^2(t) dt \leq$$

$$\tau \sum_{\text{resistors}} \phi_k(v_k(0)) + \sum_{\text{capacitors}} E_k(q_k(0)) + \sum_{\text{cells}} E_n(0), \quad (14)$$

where the right hand side is a function only of the initial conditions. Thus (5) holds. ∎

Note that Thm. 2, as it is stated, applies only to networks in which the voltage source waveform of each cell's Thevénin equivalent circuit is identically zero. In practice, these voltages are generally nonzero and change with time. Yet a necessary condition for design is that the circuit be stable for *constant* Thevénin voltages (which would result from a constant light input). If this condition is met, then the effect of time variation can be thought of as an issue separate from stability and related to the convergence rate of the network towards a "time-dependent equilibrium point." Thus, it is appropriate to extend Thm. 2 to include the case of cells that have arbitrary but constant Thevénin voltages. This can be done simply by requiring the resistor curves to satisfy the sector condition i) of the theorem about all possible equilibrium points. Even if there is no known restriction on the set of equilibrium points, the sector condition will be satisfied at every equilibrium point if all the g_k's are non-decreasing differentiable functions with bounded slope.

IV. Concluding Remarks

The design criteria presented here are simple and practical, though at present their validity is restricted to linear models of the cells. There are several areas of further work to be pursued, one of which is an analysis of the cell that includes amplifier clipping effects. Others include the synthesis of a compensator for the cell, an extension of the nonlinear result to include impedance multipliers other than the Popov operator, a bound on the network settling time when the optical input is constant, and a bound on the L_2 norm of the resistor and capacitor currents in terms of the L_2 norm of the Thevénin equivalent cell voltage waveforms when the optical input is time-varying.

ACKNOWLEDGEMENT

We sincerely thank Professor Carver Mead of Cal Tech for encouraging this work, which was supported by Defense Advanced Research Projects Agency (DARPA) Contract No. N00014-87-K-0825 and National Science Foundation (NSF) Contract No. MIP-8814612.

REFERENCES

1. F.S. Werblin, "The Control of Sensitivity in the Retina," *Scientific American*, vol. 228, no. 1, Jan. 1983, pp. 70–79.

2. T. Kohonen, **Self-Organization and Associative Memory**, (vol. 8 in the Springer Series in Information Sciences), Springer Verlag, New York, 1984.

3. C.A. Mead and M.A. Mahowald, "A Silicon Model of Early Visual Processing," *Neural Networks*, vol. 1, no. 1, Jan. 1988, pp. 91–97.

4. C.A. Mead, **Analog VLSI and Neural Systems**, Addison-Wesley, to appear in 1988.

5. J. Hutchinson, C. Koch, J. Luo and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks," *Computer*, March 1988.

6. C.A. Mead, personal communication.

7. B.D.O. Anderson, and S. Vongpanitlerd, **Network Analysis and Synthesis – A Modern Systems Theory Appraoch**, Prentice-Hall, Englewood Cliffs, NJ, 1973.

8. M. Vidyasagar, **Nonlinear Systems Analysis**, Prentice-Hall, Englewood Cliffs, NJ, 1978, pp. 211–217.

9. C. Desoer and M. Vidyasagar, **Feedback Systems: Input-Output Properties**, Academic Press, New York, 1975.

10. J.J. Hopfield, "Neurons with Graded Response have Collective Computational Properties Like Those of Two-state Neurons," *Proc. Nat'l. Acad. Sci.*, USA, vol. 81, May 1984, pp. 3088-3092.

11. L.O. Chua, C.A. Desoer and E.S. Kuh, **Linear and Nonlinear Circuits**, McGraw-Hill, 1987, sect. 4.3.

12. P. Penfield, Jr., R. Spence, and S. Duinker, **Tellegen's Theorem and Electrical Networks**, MIT Press, Cambridge, MA, 1970.

13. M. Marden, **Geometry of Polynomials**, American Mathematical Society, Providence, RI, no. 3, 1985, pp. 4–5.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-480
October 1988

A MIXED FREQUENCY-TIME APPROACH FOR DISTORTION ANALYSIS OF SWITCHING FILTER CIRCUITS

K. Kundert, J. White, A. Sangiovanni-Vincentelli

Abstract

Designers of switching filter circuits are often interested in steady-state and intermodulation distortion due to both static effects, such as nonlinearities in the capacitors, and dynamic effects, such as the charge injection during MOS transistor switching or slow operational amplifier settling. Steady-state distortion can be computed using the circuit simulation program SPICE, but this approach is computationally very expensive. Specialized programs for switched capacitor filters can be used to rapidly compute steady-state distortion, but do not consider dynamic effects. In this paper we present a new mixed frequency-time approach for computing both steady-state and intermodulation distortion. The method is both computationally efficient and includes both static and dynamic distortion sources. The method has been implemented in a C program, *Nitswit*, and results from several examples are presented.

Microsystems Research Center Room 39-321 Massachusetts Institute of Technology Cambridge Massachusetts 02139 Telephone (617) 253-8138

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-478
October 1988

EVALUATING THE PERFORMANCE OF SOFTWARE CACHE COHERENCE

Susan Owicki and Anant Agarwal

Abstract

In a *shared-memory multiprocessor* with private caches, cached copies of a data item must be kept consistent. This is called cache coherence. Both hardware and software coherence schemes have been proposed. Software techniques are attractive because they avoid hardware complexity and can be used with any processor-memory interconnection. This paper presents an analytical model of the performance of two software coherence schemes and, for comparison, snoopy-cache hardware. The model is validated against address traces from a bus-based multiprocessor. The behavior of the coherence schemes under various workloads is compared, and their sensitivity to variations in workload parameters is assessed. The analysis shows that the performance of software schemes is critically determined by certain parameters of the workload: the proportion of data accesses, the fraction of shared references, and the number of times a shared block is accessed before it is purged from the cache. Snoopy caches are more resilient to variations in these parameters. Thus when evaluating a software scheme as a design alternative, it is essential to consider the characteristics of the expected workload. The performance of the two software schemes with a multistage interconnection network is also evaluated, and it is determined that both scale well.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-491
December 1988

The J-Machine: System Support for Actors

William J. Dally

Abstract

The J-Machine in concert with its operating system kernel, JOSS, provides low-overhead system services to support actor programming systems. The J-Machine is not specialized to actor systems; instead, it provides primitive mechanisms for communication, synchronization, and translation. Communication mechanisms are provided that permit a node to send a message to any other node in the machine in $< 2\mu s$. On message arrival, a task is created and dispatched in $< 1\mu s$. A translation mechanism supports a global virtual address space. These mechanisms efficiently support most proposed models of concurrent computation. The hardware is an ensemble of up to 65,536 nodes each containing a 36-bit processor, 4K 36-bit words of memory, and a router. The nodes are connected by a high-speed 3-D mesh network. This design was chosen to make the most efficient use of available chip and board area.

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-492
December 1988

Universal Packet Routing Algorithms

Tom Leighton, Bruce Maggs, and Satish Rao

Abstract

In this paper we examine the packet routing problem in a network independent context. Our goal is to devise a strategy for routing that works well for a wide variety of networks. To achieve this goal, we partition the routing problem into two stages: a path selection stage and a scheduling stage.

In the first stage we find paths for the packets with small maximum distance, d, and small maximum congestion, c. Once the paths are fixed, both are lower bounds on the time required to deliver the packets. In the second stage we find a schedule for the movement of each packet along its path so that no two packets traverse the same edge at the same time, and so that the total time and maximum queue size required to route all of the packets to their destinations are minimized. For many graphs, the first stage is easy - we simply use randomized intermediate destinations as suggested by Valiant. The second stage is more challenging, however, and is the focus of this paper. Our results include:

1. a proof that there is a schedule of length $O(c+d)$ requiring only constant size queues for any set of paths with distance d and congestion c,

2. a Randomized on-line algorithm for routing any set of N "leveled" paths on a bounded-degree network in $O(c+d+\log N)$ steps using constant size queues,

3. the first on-line algorithm for routing N-packets in the N-node shuffle-exchange graph in $O(\log N)$ steps using constant size queues, and

4. the first constructions of area and volume-universal networks requiring only $O(\log N)$ slow-down.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-493
November 1988

Criteria for Robust Stability in a Class of Lateral Inhibition Networks Coupled Through Resistive Grids

John L. Wyatt, Jr. and David L. Standley

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid to interconnect active elements. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports on criteria that guarantee these and certain other systems will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values ot parasitic elements may be unknown. The method is based on a rigorous, somewhat novel mathematical analysis using Tellegen's theorem from electrical circuits and the idea of a Popov multiplier from control theory. The criteria are *local* in that no overall analysis of the interconnected system is required for their use, *empirical* in that they involve only measurable frequency response data on the individual cells, and *robust* in that they are insensitive to network topology and to unmodelled parasitic resistances and capacitances in the interconnect network. Certain results are robust in the additional sense that specified *nonlinear* elements in the grid do not affect the stability criteria. The results are designed to be applicable, with further development, to complex and incompletely modelled living neural systems.

Microsystems Research Center Room 39-321 Massachusetts Institute of Technology Cambridge Massachusetts 02139 Telephone (617) 253-8138

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-494
November 1988

Stability Criterion for Lateral Inhibition and Related Networks that is Robust in the Presence of Integrated Circuit Parasitics

John L. Wyatt, Jr. and David L. Standley

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values of parasitic elements may be unknown. The method is based on a mathematical analysis using Tellegen's theorem and the Popov criterion. The criteria are *local* in the sense that no overall analysis of the interconnected system is required for their use, *empirical* in the sense that they involve only measurable frequency response data on the individual cells, and *robust* in the sense that they are not affected by unmodelled parasitic resistances and capacitances in the interconnect network.

Easily Testable PLA-based Finite State Machines

Srinivas Devadas
Department of Electrical Engincering and Computer Science
Massachusetts Institute of Technology, Cambridge

Hi-Keung Tony Ma and A. Richard Newton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

In this paper, we outline a synthesis procedure, which beginning from a State Transition Graph description of a sequential machine, produces an optimized **easily testable** PLA-based logic implementation.

Previous approaches to synthesizing easily testable sequential machines have concentrated on the stuck-at fault model. For PLAs, an extended fault model called the crosspoint fault model is used. In this paper, we propose a procedure of **constrained state assignment and logic optimization** which guarantees testability for all combinationally irredundant crosspoint faults in a PLA-based finite state machine. No direct access to the flip-flops is required. The test sequences to detect these faults can be obtained using combinational test generation techniques alone. This procedure thus represents an alternative to a Scan Design methodology. We present results which illustrate the efficacy of this procedure – the area/performance penalties in return for easy testability are small.

1 Introduction

Test generation for sequential circuits has long been recognized as a difficult task [4]. Several approaches [3] [18] [15] [14] [17] [19] have been taken in the past to solve the problem of test generation for sequential circuits. They are either extensions to the classical D-Algorithm or based on random techniques [18] [17]. When the number of states of the circuit is large and the tests demand long input sequences, they can be quite ineffective for test generation.

For sequential circuits, design for testability has been a synonym for the use of full Scan Design techniques, such as the LSSD approach [10] pioneered by IBM. This method converts the difficult problem of testing sequential circuits, into a much easier one, that of testing a combinational circuit. However, there are cases where the area and timing penalty associated with LSSD techniques are not acceptable to designers.

Logic synthesis and minimization techniques can, in principle, ensure fully and easily testable combinational and sequential circuit designs. In [1], a synthesis procedure which guaranteed fully testable irredundant combinational logic circuits was proposed. In [8], a procedure which produced a fully and easily testable logic-level sequential machines from State Transition Graph descriptions was proposed. The work in [8] showed that state assignment has a profound effect on the testability of a sequential machine. Recently, an optimal synthesis procedure, that guarantees full non-scan testability under the stuck-at fault model, with no associated area or performance penalty has been proposed [6].

Programmable Logic Arrays (PLAs) are used extensively in the design of complex VLSI systems. Sequential functions can be realized very efficiently by adding feedback registers to the PLA. Numerous programs for the optimal synthesis of PLA-based finite state machines have been developed (e.g. [16], [7]). Test generation and design-for-testability techniques for PLA structures have been active areas of research.

Due to a PLA's dense layout, PLA faults other than conventional stuck-at faults can occur easily and must be modeled. An extended model, the *crosspoint fault* model, has been proposed in [5] and [12]. The crosspoint-oriented test set covers many of the frequently occurring physical faults, including shorts between lines. Several PLA test generation techniques aimed at the crosspoint fault model have been proposed (e.g. [13], [9]). In particular, an exact and efficient technique which guarantees maximum fault coverage and identification of all redundant faults was proposed in [20].

Design-for-testability techniques (e.g. [11]) for PLAs require controllability of all inputs and observability of all outputs of the PLA. Synthesis approaches to producing easily testable sequential machines, without requiring direct access to the inputs/outputs of the circuit's memory elements, have not been aimed at the crosspoint fault

Temporal, Processor, and Spatial Locality in Multiprocessor Memory References*

Anant Agarwal
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Anoop Gupta
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Abstract

The performance of cache-coherent multiprocessors is strongly influenced by locality in the memory reference behavior of parallel applications. While the notions of temporal and spatial locality in uniprocessor memory references are well understood, the corresponding notions of locality in multiprocessors and their impact on multiprocessor cache behavior are not clear. A locality model suitable for multiprocessor cache evaluation is derived by viewing memory references as streams of processor identifiers directed at specific cache/memory blocks. This viewpoint differs from the traditional uniprocessor approach that uses streams of addresses to different blocks emanating from specific processors. Our view is based on the intuition that cache coherence traffic in multiprocessors is largely determined by the number of processors accessing a location, the frequency with which they access the location, and the sequence in which their accesses occur. The specific locations accessed by each processor, the time order of access to different locations, and the size of the working set play a smaller role in determining the cache coherence traffic, although they still influence intrinsic cache performance. Looking at traces from the viewpoint of a memory block leads to a new notion of reference locality for multiprocessors, called processor locality. In this paper, we study the temporal, spatial, and processor locality in the memory reference patterns of three parallel applications. Based on the observed locality, we then reflect on the expected cache behavior of the three applications.

1 Introduction

Multiprocessors often use caches to reduce their network bandwidth requirements. Caches retain recently accessed data so that repeat references to this data in the near future and will not require network traversals. Repeated access to the same data in a given interval of time is the property of temporal locality of memory references and has been well studied in single processor systems [1, 2]. Spatial locality of memory references is another related property of memory references that places a high probability of access to data close to previously accessed data. Again, this property of single processor programs has been widely observed. The viability of cache-coherent multiprocessors is strongly predicated on whether the multiprocessor caches can exploit locality of memory referencing.

*Preliminary results of this study were reported in Sigmetrics 1988.

Approaches to Multi-Level Sequential Logic Synthesis

Srinivas Devadas

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

Abstract

In this paper, we present approaches to multi-level sequential logic synthesis – algorithms and techniques for *the area and performance optimization of interconnected finite state machine descriptions.*

Interacting finite state machines are common in industrial chip designs. While optimization techniques for single finite state machines are relatively well developed, the problem of optimization across latch boundaries has received much less attention. Techniques to optimize pipelined combinational logic so as to improve area/throughput have been proposed. However, logic *cannot* be straightforwardly migrated across latch boundaries when the basic blocks are sequential rather than combinational circuits.

We present new techniques for the exploitation of *sequential don't cares* in arbitrary, interconnected sequential machine structures. Exploiting these don't care sequences can result in significant improvements in area and performance. We address the problem of migrating logic across state machine boundaries so as to make particular machines less complex at the possible expense of making others more complex. This can be useful from both an area and performance point of view. We present new optimization algorithms that *incrementally* modify state machine structures across latch boundaries. We discuss the use of more global state machine decomposition and factorization algorithms for area optimization. Finally, we present experimental results using these algorithms on sequential circuits.

1 Introduction

Interacting finite state machines (FSMs) are common in chips being designed today. The advantages of a hierarchical, distributed-style specification and realization are many. While the terminal behavior of any set of interconnected sequential circuits can be modeled and/or realized by a lumped circuit, the former can be considerably more compact, as well as being easy to understand and manipulate.

The disadvantages of this form of specification from a CAD point of view are that sequential logic synthesis algorithms are generally restricted to operate on lumped circuits. State assignment algorithms (e.g. [1], [8], [3]), for instance, almost exclusively operate on single finite state machines. Given a set of interacting machines represented by State Transition Graphs, algorithms that encode the internal states of the machines, *taking into account their interactions*, do not exist to date. If indeed, the machines are encoded separately, disregarding their interconnectivity, a sub-optimal state assignment can result (and generally does).

Traditionally, the decomposition of an initial circuit specification into smaller, interacting sequential circuits has been performed by the logic designer. Once a decomposition has been performed, it is almost never changed and logic synthesis tools operate on separate logic blocks independently. Unfortunately, there are no guarantees regarding the quality of the initial decomposition, in terms of minimality of communication between the machines and/or complexities of the individual machines. There exist automatic techniques that can decompose lumped sequential circuits into smaller, interacting ones (e.g. [5]). These techniques are limited in the topology of interconnections that can be achieved and severely limited in their capabilities of handling circuits of large size. Flattening the initial distributed specification can result in a *very* large lumped circuit.

Efficient and flexible algorithms for re-partitioning interacting sequential circuits for area and performance optimization have not been proposed in the past. Work has been done in re-partitioning pipelined combinational logic stages (e.g. [6]). There is no restriction on migrating logic across latch boundaries when the basic blocks are combinational, provided the latches are not observable – the functionality of the circuit is unchanged by moving say, one gate from before to after a latch. However, when sequential circuits are interconnected, as shown in Figure 1, one *cannot* arbitrarily move logic across pipeline latch boundaries (We refer to flip-flops that store state as state latches and flip-flops that store intermediate values as pipeline latches). The functionality and terminal behavior of the circuit will be changed, even though the latches are not observable.

One wishes to be able to migrate logic across pipeline latch boundaries for several reasons. The duration of the system clock has to be greater than the longest path between any two pipeline stages. If a machine, A, is significantly more complex than another machine B, the critical path/system clock may be unnecessarily long. The clock cycle could be shortened by making A less complex at the possible expense of making B more complex. In the best case, the complexities of both A and B would decrease.

Another very important issue is the specification and exploitation of don't cares in interconnected FSM descriptions. For example, in Figure 1, certain binary combinations may never appear at the set of latches $L1$. This will correspond to an incompletely specified machine B. These don't cares can be exploited using standard state minimization strategies [9]. A more complicated form of don't care, referred to here as a *se-*

Area-Universal Networks

Ronald I. Greenberg
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
rig@theory.lcs.mit.edu

(extended abstract)
January 24, 1989

An area-universal network is one which can efficiently simulate any other network of comparable area. This paper extends previous results on area-universal networks in several ways. First, it considers the size (amount of attached memory) of processors comprising the networks being compared. It shows that an appropriate universal network of area $\Theta(A)$ built from processors of size $\lg A$ requires only $O(\lg^2 A)$ slowdown *in bit-times* to simulate any network of area A, without any restriction on processor size or number of processors in the competing network. Furthermore, the universal network can be designed so that any message traversing a path of length d in the competing network need follow a path of only $O(d + \lg A)$ length in the universal network. Thus, the results are almost entirely insensitive to removal of the unit wire delay assumption used in previous work. This paper also derives upper bounds on the slowdown required by a universal network to simulate a network of larger area and shows that all of the simulation results are valid even without the usual assumption that computation and communication of the competing network proceed in separate phases.

1 Introduction

This paper provides several advances in the search for the best way to make use of a fixed amount of physical space when building a general-purpose parallel computer. The focus on space consumed by both processors and interconnect represents an attempt to better measure real-world costs than to merely count the number of processors. The results of this paper are stated in terms of area required under standard two-dimensional VLSI modeling assumptions. The extension to three-dimensions is fairly straightforward using the ideas in [2].

The notion of a routing network which is universal for a given amount of physical space was introduced by Leiserson in [5]. That paper introduces a class of routing networks referred to as fat-trees and shows that an appropriate n-processor network from this class can simulate (off-line) any other routing network connecting the same processors and occupying the same volume, with only $O(\lg^3 n)$ factor degradation in the time required. A slight restriction on the number of processors in the competing network was required, because Leiserson's fat-trees used area slightly more than linear in the number of processors.

The approach to proving fat-trees universal is twofold. First it is shown that any competing network can be mapped to a fat-tree without placing too great a communications load on any of the communication

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-0825, by the Office of Naval Research under Contract N00014-86-K-0593, and by a Fannie and John Hertz Foundation Fellowship.

A Digital Model for Level-Clocked Circuitry

by
Alexander Toichi Ishii

Submitted to the Department of
Electrical Engineering and Computer Science
on August 8, 1988
in Partial Fulfillment of the
Requirements of the Degree of

Master of Science
in
Electrical Engineering and Computer Science

Abstract

This thesis presents the formal background for a mathematical model for level-clocked circuitry, in which latches are controlled by the levels (high or low) of clock signals rather than transitions (edges) of the clocks. Such level-clocked circuits are frequently used in MOS VLSI design. Our model maps continuous data-domains, such as voltage, into discrete, or *digital*, data domains, while retaining a continuous notion of time. A level-clocked circuit is represented as a graph $G = (V, E)$, where V consists of digital components—latches and functional elements—and E represents inter-component connections.

The majority of this thesis concentrates on developing lemmas and theorems that can serve as a set of "axioms" when analyzing algorithms based on the model. Key axioms include the fact that circuits in our model generate only well defined digital signals, and the fact that components in our model support and accurately handle the "undefined" values that electrical signals must take on when they make a transition between valid logic levels. In order to facilitate proofs for circuit properties, the class of *computational predicates* is defined. A circuit property can be proved by simply casting the property as a computational predicate.

Thesis Supervisor: Professor Charles E. Leiserson
Title: Associate Professor

[0]This research represents joint work with Charles E. Leiserson, and is supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825. Charles Leiserson is supported in part by an NSF Presidential Young Investigator Award with matching funds provided by AT&T Bell Laboratories and Xerox Corporation.

A RECONFIGURABLE ARITHMETIC PROCESSOR

by

James Alexander Stuart Fiske

Submitted to the
Department of Electrical Engineering and Computer Science
on December 16 in partial fulfillment of
the requirements for the Degree of Master of Science in
Electrical Engineering and Computer Science

Abstract

Achieving high rates of floating-point computation is one of the primary goals of many computer designs. Many high speed floating-point datapaths have been designed in order to address this problem. However, conventional designs often neglect the real problem in achieving high performance floating-point: providing the necessary I/O bandwidth to keep the high speed datapaths busy.

The Reconfigurable Arithmetic Processor (RAP) is an arithmetic processing node for a message-passing, MIMD concurrent computer. Its datapath is designed to sustain high rates of floating-point operations, while requiring only a fraction of the I/O bandwidth required by a conventional floating-point datapath. The RAP incorporates on one chip eight 4-bit serial, 64 bit floating-point arithmetic units connected by a switching network. By sequencing the switch through different patterns, the RAP chip calculates complete arithmetic formulas. By chaining together its arithmetic units the RAP eliminates the I/O bandwidth associated with storing and retrieving intermediate results, and reduces the amount of off chip data transfer.

This Thesis describes and evaluates the RAP architecture. It presents two important aspects of the chip design: the control logic design, and the schematic level design of the RAP datapath. The RAP datapath design includes the design of two 4-bit serial floating-point units: an adder/subtractor unit and a multiplier unit. In order to use the RAP datapath, a compiler is developed that takes as input a list of mathematical expressions, and outputs a series of switch configurations to be used by the RAP to do the calculation.

On 23 benchmark problems, the RAP reduced both the on chip and off chip bandwidth requirements by an average of 64%, when compared the bandwidth required by a conventional arithmetic chip that does not exploit locality. Average floating-point performance is 3.40 Millions of Floating-point operations per second (MFlops).

Thesis Supervisor: William J. Dally
Title: Assistant Professor of Electrical Engineering and Computer Science

Keywords: *Floating-Point*, B[illegible]dwidth, Serial Arithmetic, Locality, J-Machine.