NEW YORK UNIVERSITY
Ultracomputer Research Laboratory

Courant Institute of Mathematical Sciences
251 Mercer Street
New York, NY 10012

ULTRACOMPUTER NOTE ABSTRACTS
Revised: January, 1990

UC. Jack Schwartz, ''Ultracomputers''

A class of parallel processors potentially involving thousands of individual processing elements is described. The architecture is based on the perfect shuffle connection and has two favorable characteristics: (1) Each processor communicates with a fixed number of other processors. (2) Important communication functions can be accomplished in time proportional to the logarithm of the number of processors. A number of basic algorithms for these ''ultracomputers'' are presented, and physical design considerations are discussed in a preliminary fashion.

1. Lambert Meertens, ''Bitonic Sort on Ultracomputers''

Batcher's *bitonic sort* is a sorting network, capable of sorting n inputs in $\Theta((\log n)^2)$ stages. When adapted to conventional computers, it gives rise to an algorithm that runs in time $\Theta(n(\log n)^2)$. The method can also be adapted to *ultracomputers* to exploit their high degree of parallelism. The resulting algorithm will take time $\Theta((\log N)^2)$ for ultracomputers of ''size'' N. The implicit constant factor is low, so that even for moderate values of N the ultracomputer architecture performs faster than the $\Theta(N \log N)$ time conventional architecture can achieve. The purpose of this note is to describe the adapted algorithm. After some preliminaries a first version of the algorithm is given whose correctness is easily shown. Next, this algorithm is transformed to make it suitable for an ultracomputer.

2. Lambert Meertens, ''Recurrent Thought on Ultracomputers Programming Style''

Ultracomputers are assemblages of processors that are able to operate concurrently and can exchange data through communication lines in, say, one cycle of operation. For physical reasons, the fan in/out of the processors must be limited. This imposes restrictions on the possible communication schemes. In order to have the ultracomputer operate efficiently as a whole, it is desirable that arbitrary exchanges of information between the processors can be effected in a small number of data shifts.

If a really huge ultracomputer is built, it would be nice if it could be constructed by coupling smaller ultracomputers, which in turn are assembled from still smaller ultracomputers, and so on. It will be shown that the latter desire conflicts to a certain extent with the earlier one.

3. Jack Schwartz, ''Preliminary Thought on Ultracomputers Programming Style''

This note describes a simple low-level language which can be used to write ultracomputer programs of the synchronous type, i.e. programs oriented toward the coordinated use of all the processors of an ultracomputer.

4. Jack Schwartz, ''A Remark on Nearly Planar Embeddings of Small Ultracomputers''

For efficient VLSI realizations of ultracomputers, planar or nearly planar realizations of the shuffle interconnection are desirable. In the N = * case this is easily done. In the N = 16 case a nearly planar layout is given and shown to be optimal.

5. Jack Schwartz, ‘‘The Burroughs FMP Machine’’

This note comments on various interesting points which appear in the Burroughs Corporation final technical report NAS2-9897, March 1979, entitled *Numerical Aerodynamic Simulation Facility Feasibility Study*. The points covered include: The data movement algorithm of the interconnection network, physical structure of the interconnection network and timings, programming style, applications benchmarked, and reliability considerations.

6. Clyde P. Kruskal and Larry Rudolph, ‘‘Observations Concerning Multidimensional Ultracomputers’’

Schwartz introduced the idea of an ultracomputer and reviewed various basic algorithms for such an ensemble of processors containing ‘‘shuffle’’ interconnections. Later, Harrison and Kalos introduced the idea of an N-dimensional ultracomputer (N-D UC), which is well suited for implementing many algorithms. The main result of this note is that an N-D UC can simulate the right/left and shuffle/unshuffle connections of its 1-dimensional counterpart in N and 2N steps, respectively. Thus, for each N, an N-D UC is asymptotically as fast as a 1-D UC of the same size.

7. Allan Gottlieb and Clyde P. Kruskal, ‘‘A Data-Motion Algorithm’’

We desire to permute N items $w_0,...,w_{N-1}$, in an ultracomputer containing P processing elements (PEs), $PE_0,...,PE_{P-1}$. Under the assumption that N=P and that $w_i \varepsilon PE_i$, Schwartz gives the following worst case analyses: The static permutation algorithm requires 4 log P - 3 data communication steps, and the dynamic permutation problem requires $\theta((\log P)^2)$ data communications steps. It is easily seen that for both algorithms the average case behavior closely approximates the worst case.

Here we present a data motion algorithm oriented toward average case rather than worst case performance, and supply an argument suggesting that the average number of data communication steps required is approximately 3 log P.

8. Allan Gottlieb, ‘‘A Remark on the Planarity of the Shuffle-Exchange Networks of Sizes 16 and 32’’

The shuffle-exchange network omits some of the connections that are present in a full ultracomputer. This note demonstrates that a 16 processor network of this sort can be imbedded in the plane without crossings. In addition, we show that if just 6 connections are allowed to pass through the ‘‘active’’ area occupied by another processor then the 32 processor case is also planar. Connections of this sort may be feasible in a VLSI technology.

9. Allan Gottlieb, ‘‘Another Remark on the Planarity of Shuffle-Exchange Networks of Sizes 16 and 32’’

Rudolph presented an example of a 32 processor shuffle-exchange network in which 6 connections pass through the area occupied by another processor. In this note we reduce the number of intersections from 6 to 4. Since none of the exchange connections pass through a processor, the exchange operation can be performed in one cycle. The shuffle operation requires 2 cycles.

10. Allan Gottlieb, ‘‘Plus – A PL/I Based Ultracomputer Simulator, I’’

This note describes a (presently implemented) ultracomputer emulator called PLUS which uses the multitasking and preprocessing features of PL/I to support a recursive ultracomputer programming style. Since the emulation to be described is written in in PL/I, the powerful debugging features of the PL/I checkout compiler, as well as PL/I’s separate compilation facility are available. This note also describes the emulation system and furnish a ‘‘User’s Guide’’. In a subsequent part II we will discuss the system’s implementation and prove both correctness and freedom from deadlock.

11. Allan Gottlieb and Clyde P. Kruskal, ‘‘Supersaturated Ultracomputer Algorithms’’

For a wide class of problems, we obtain lower bounds for algorithms executed on certain parallel processors. These bounds show that for sufficiently large problems many known algorithms are optimal. The central result of the paper is the following sharper lower bound for permutation algorithms. Any permutation algorithm for N data items on a P processor parallel machine without shared memory requires time on the order of $(N/P) \log_k P$ where K is the maximum number of processors directly connected to a single processor. In particular, a speedup on the order of P is impossible if K is bounded.

12. Allan Gottlieb, ‘‘WASHCLOTH – The Logical Successor to SOAPSUDS’’

Schwartz has proposed the ‘‘paracomputer’’ model of parallel computation whose performance more realistic architectures could try to approach. This note describes the WASHCLOTH paracomputer simulator and is organized as follows: Section II reviews the paracomputer model; section III presents the WASHCLOTH design and the machine operations provided; section IV describes the FORTRAN interface; and section V is a users' guide to WASHCLOTH.

13. Allan Gottlieb, ‘‘MOP – A (Minimal) Multiprocessor Operating System Extending WASHCLOTH’’

This note describes MOP, a skeletal multiprocessor operating system for the WASHCLOTH virtual machine. The note is organized as follows: Chapter II describes the WASHCLOTH virtual machine, chapter III presents the MOP user interface, and chapter IV proposes an initial implementation.

14. Allan Gottlieb, ‘‘PLUS: A PL/I Based Ultracomputer Simulator, II’’

Part 1 of this two part report introduced a (presently implemented) ultracomputer simulator called PLUS which uses the multitasking and preprocessing features of PL/I to support a recursive ultracomputer programming style. Since the simulation is written in PL/I, the powerful debugging features of the PL/I checkout compiler, as well as PL/I's separate compilation facility are available. In this note we discuss the system's implementation and prove both correctness and freedom from deadlock.

15. Allan Gottlieb and Clyde P. Kruskal, ‘‘MULT – A Multitasking Ultracomputer Language with Timing, I & II’’

[UC] introduced the idea of an ultracomputer, a parallel processor based on the ‘‘shuffle’’ connections; reviewed various basic algorithms for such an ensemble of processors; and analyzed their asymptotic time complexities. However, one will want to know exact, rather than merely asymptotic, efficiencies when designing and using an ultracomputer. Moreover, some algorithms are difficult to analyze analytically (e.g. linear programming and backtracking) and only actual practice gives us a feeling for their efficiencies. The MULT (*M*ultiprocessor *U*ltracomputer *L*anguage with *T*iming) system described in this note is an attempt to deal with these issues by timing simulated executions of ultracomputer programs.

16. Allan Gottlieb, B.D. Lubachevsky, and Larry Rudolph, ‘‘Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors’’

In this paper we implement several basic operating system primitives by using a ‘‘replace-add’’ operation, which can supersede the standard ‘‘test and set’’, and which appears to be a universal primitive for efficiently coordinating large numbers of independently acting sequential processors. We also present a hardware implementation of replace-add that permits multiple replace-adds to be processed nearly as efficiently as loads and stores. Moreover, the crucial special case of concurrent replace-adds updating the same variable is handled particularly well: If every PE simultaneously addresses a replace-add at the same variable, all these requests are satisfied in the time required to process just one request.

17. Allan Gottlieb, ‘‘Comments on Concurrent Search and Insertion in AVL Trees’’

Ellis’s concurrent AVL insertion algorithm is discussed in this correspondence. We note that obtaining a block of storage for the new AVL leaf may become a serial bottleneck for the entire insertion algorithm. We indicate a potential solution and refer the reader to another paper in which the full details are given.

18. Charles S. Peskin and Olof B. Widlund, ‘‘Remarks on Efficient Numerical Algorithms for Ultracomputers’’

This note gives an outline of a few numerical algorithms for some important problems arising in continuum mechanics which can take full advantage of the special features of ultracomputers. We discuss fast Poisson solvers, capacitance matrix or imbedding methods for special elliptic problems on general bounded domains and a finite difference approximation to the time dependent three dimensional Navier Stokes equations. We show that algorithms exist which are within a logarithmic factor of achieving optimal speedup and which are relatively easy to implement. We conclude with a few general comments on the parallel solution of linear systems of algebraic equations.

19. Charles S. Peskin, ‘‘Ultracomputer Implementation of Odd-Even Reduction’’

The Odd-Even Reduction Algorithm has been widely used to solve tridiagonal systems of equations in serial computers, especially in the context of fast Poisson-solvers. The parallel form of the algorithm is also discussed by Heller. The purpose of this note is to show that the structure of the algorithm is just right for implementation on an ultracomputer.

20. Charles S. Peskin, ‘‘A Comparison of Ultracomputer Architecture And Lattice Architecture for Problems on Lattices’’

We conclude that for most numerical problems on lattices, an ultracomputer will be faster than a lattice machine, despite the fact that the lattice machine seems to have precisely the right architecture for the problem. There is a special class of problems where the lattice machine is faster, but only by a factor of log N. In either case, the ultracomputer will be a more flexible machine, adaptable to a variety of problems without rewiring.

21. Allan Gottlieb, ‘‘WASHCLOTH 81’’

This note describes several enhancements that have been made to the WASHCLOTH simulator.

22. Norman Rushfield, ‘‘Atmospheric Computations on Highly Parallel MIMD Computers’’

A FORTRAN program representing a two-dimensional vertical plane model of the atmosphere has been modified to make use of the WASHCLOTH simulator of a parallel processor. To determine the suitability of this code for execution on very large scale parallel computers of the proposed paracomputer class, we analyzed the performance of three codes: the original code; code 1 as modified to run under the WASHCLOTH parallel processor simulator; and a fully parallel code that partitions work among multiple PEs.

23. David Korn, ‘‘Converting Scientific Codes to Run Under the WASHCLOTH Simulator’’

This note describes a set of subroutines that have been written to aid in preparing scientific FORTRAN code for WASHCLOTH simulation. In addition to making code conversion simpler, these routines count machine instructions spent waiting for processor synchronizations. Finally, the SOFTSOAP package that enables FORTRAN code prepared for WASHCLOTH simulation to run with 1 processor in a simple, efficient manner is presented.

24. David Korn, ‘‘Timing Analysis for Scientific Codes Run Under WASHCLOTH Simulation’’

This note details the timing measurements made for scientific programs run under the WASHCLOTH simulator. Processing time, as a function of the number of processing elements and the problem rate, is analyzed. The aim is to predict how the number of processors that can be used efficiently varies as a function of the problem size. In addition, the analysis sheds light on the problem of designing and coding efficient programs.

25. Allan Gottlieb and Jack Schwartz, ‘‘Networks and Algorithms for Very Large Scale Parallel Computation’’

After discussing interconnection networks in general, this report illustrates the operation of three well-known examples: the cube, the shuffle, and the cube connected cycle. The important ‘‘dynamic permutation’’ problem (similar to the problem of routing memory requests to the appropriate memory module) is then described and analyzed. These results are particularly applicable to machine designs similar to Burrough’s FMP. The paracomputer model of computation and the replace-add primitive are reviewed, and hardware issues are discussed. We show how this permits queues to be managed in a highly parallel manner, which in turn gives a concurrent implementation of a dataflow language.

26. Clyde P. Kruskal, ‘‘Supersaturated Paracomputer Algorithms’’

This paper describes the paracomputer model of parallel computation and presents paracomputer algorithms for solving several important problems including sorting. A paracomputer is essentially a traditional shared memory machine augmented with an extra primative – the ‘‘replace-add’’. We show that this primative enhances the model by exhibiting algorithms that are faster that any possible algorithm for solving the same problem on a shared memory machine. In addition, several of the algorithms are faster than any known algorithm for solving the same problem on a shared memory machine.

27. Malvin H. Kalos, Gabi Leshem, and B.D. Lubachevsky, ‘‘Molecular Simulations of Equilibrium Properties’’

A Monte Carlo algorithm for sampling representative configurations of a classical many-body system in equilibrium was implemented as a program for an asynchronous shared memory parallel computer. The algorithm is the iterative repetition of a cycle involving serial initialization and termination phases and a dominant computation in the middle that is suitable for parallelization. The efficiency of the parallel implementation is increased by allowing more than one such a cycle to be treated concurrently. Two versions of the implementation are discussed. The first one uses only one parallel feature, basic indivisible operation Fetch-and-Add and has necessary service subroutines built in. The second one employs primitives Request, Release, and Suspend of a possible parallel operating system. Timing analysis of simulated executions predicts high efficiency of this algorithm for parallel computers that include up to thousands of processing elements.

28. Marc Snir, ‘‘‘NETSIM’ Network Simulator for the Ultracomputer’’

This note describes an extension to WASHCLOTH that introduces extra delays for references to shared variables. The simulator is not intended to yield accurate measurements of the performance of an actual machine. A precise simulation of the communication network we intend to implement would be extremely time consuming. We made several simplifying assumptions, and used some rough estimates on the relative performance of different hardware components of the system.

In section two, we present the probabilistic model used. In section three, we detail the information used to drive the simulation, and then discuss the simplifying assumption made. A summary of results obtained follows, along with a discussion of these, and suggestions for future work.

29. Marc Snir, ‘‘Lower Bounds on VLSI Implementations of Communication Networks’’

An $O(n^2)$ lower bound on the area required for VLSI implementations of permutation networks and concentrators is given. A tradeoff result for the bandwidth of packet switching networks and the area required to implement them is proven.

30. Malvin H. Kalos, ‘‘Scientific Computations on the Ultracomputer’’

This note reviews very briefly the architecture of the NYU Ultracomputer, a highly parallel, shared memory MIMD machine. The use of the replace add instruction for scheduling parallel tasks is also sketched. Finally, some experience in using the WASHCLOTH simulator to study some representative numerical programs is summarized. These include a Monte Carlo program for statistical mechanics; a program for solving the Navier-Stokes equation for blood flow; parallelization of Householder’s algorithm for reduction of a symmetric matrix to tridiagonal form; and a program for modeling atmospheric flow in two dimensions.

31. David Korn, ‘‘Timing Simulations for Elliptic PDE’s Run Under WASHCLOTH’’

Two methods for solving three dimensional elliptic partial differential equations have been modified for parallel execution on a paracomputer and run under WASHCLOTH simulation. A brief discussion of these methods and timing results under simulation are presented in this note.

32. UCN 32 IS OUT OF PRINT.

33. B.D. Lubachevsky, ‘‘Verification of Several Parallel Coordination Primitives Based on Descriptions of Their Reachability Sets’’

A method for verifying parallel programs is applied to several examples (PV-semaphore, ‘‘busy-wait’’ synchronization, ‘‘cessation of activity’’ synchronization, ‘‘readers-writers’’). The graph of all program states and state transitions is represented in a special compact form independent of the number N of processing elements. This representation aids in verifying certain correctness properties that can not be easily expressed in the form ‘‘predicate(state)’’. In each of the above mentioned examples a special ‘‘reachability tree’’ is developed whose nodes are some subsets of the set of all reachable states. The root is the initial state and moving down the tree corresponds to some processors advancing their execution. In the presented examples the size of this tree is independent of N. The notion of compact program is introduced: roughly speaking a parallel program is compact if there exists a boundary, independent of N, on time required to reach any state. Examples of non-compact programs are represented.

34. Allan Gottlieb and Clyde P. Kruskal, ‘‘Coordinating Parallel Processors: A Partial Unification’’

A variety of coordination primitives have been proposed for memory-sharing MIMD parallel processors. The most familiar of these primitives are probably test-and-set and swap. In this report, we show how a simple modification of the fetch-and-add coordination primitive recently proposed for the ‘‘NYU Ultracomputer’’ yields a hardware implementation technique that may also be used for test-and-set and swap. This technique enables one to support a wide range of operations on any network-based multiprocessor in a highly parallel manner.

35. B.D. Lubachevsky, ‘‘Review of Soviet Publications on Parallel Data Processing’’

The aim of this paper is to review the available Soviet publications on the topic of parallel data processing published during 1978-1980. (A few articles published outside of this interval are also included as they were relevant.)

36. Allan Gottlieb, ‘‘A Historical Guide to the Ultracomputer Literature’’

When reading our reports, one should note that three distinct parallel processors are discussed: the ‘‘ultracomputer’’, a message passing machine; the ‘‘paracomputer’’, an idealized shared memory machine; and the ‘‘NYU Ultracomputer’’, a network-based shared memory machine that approximates a paracomputer. Moreover, since the NYU Ultracomputer has replaced the original ultracomputer as out preferred design, our later reports often ‘‘abbreviate’’ NYU Ultracomputer as Ultracomputer. Finally, our basic coordination primitive has been called NEWVAL, replace-add, and fetch-and-add. We realize that such highly context sensitive language may be confusing and this guide is a (long overdue) attempt to help clarify the situation. In addition, by presenting a historical overview of the project I hope to group together those reports reflecting work done at roughly the same time (and hence using the same terminology).

37. B.D. Lubachevsky, ‘‘A Parallel Computer Implementation of the Ascend/Descend Types of Vector Algorithms’’

An implementation of the ASCEND/DESCEND types of vector algorithms (like FFT) is adapted to the current model of ‘‘NYU-Ultracomputer’’, a shared memory asynchronous parallel processor, including basic features of the proposed hardware and software.

38. Marc Snir, ‘‘Comments on Lens and Hypertrees – or the Perfect-Shuffle Again’’

Finkel and Solomon introduce the lens interconnection strategy, and claim it is the only one known having small diameter, fixed degree, simple routing algorithm, uniform traffic load, and redundancy. We show that the ‘‘de Bruijn’’ network is nothing less than the shuffle-exchange network of Stone in disguise.

39. Marc Snir and Jon Solworth, ‘‘The Ultraswitch – A VLSI Network Node for Parallel Processing’’

A detailed description is given of the network connecting processors (PE’s) to memory modules (MM’s) in the NYU Ultracomputer, and of the switches used to realize this network

The network is a packet switched multi-stage shuffle-exchange network. All the switches are identical, and built of a small number of chips: two monolithic integrated circuits and one small arbiter chip.

In addition to routing, these chips can queue information when the network is congested. A third function is the combining of like requests to the same memory address. This capability allows efficient concurrent access by many PE’s to the same location in memory, and effectively distributes the support of synchronization primitives (Fetch-and-Add) over the entire network.

The logic of each individual switch is pipelined, and several systolic structures are used to achieve a high throughput.

40. UCN 40 HAS BEEN SUPERSEDED BY UCN 100.

41. Clyde P. Kruskal and Marc Snir, ‘‘Some Results on Multistage Interconnection Networks for Multiprocessors’’

This paper studies the performance of unbuffered and buffered, packet-switching, multistage interconnection networks. We begin by reviewing the definition of banyan networks and introducing some generalizations of them. We then present an asymptotic analysis of the performance of unbuffered banyan networks, thereby solving a problem left open by Patel. We analyze the performance of the unbuffered generalized banyan networks, and compare networks with approximately equivalent hardware complexity. Finally, we analyze the performance of buffered banyan networks and again compare networks with approximately equivalent hardware complexity.

42. B.D. Lubachevsky, ‘‘Parallelizing an Algorithm of Charles S. Peskin for Describing Incompressible Flow of a Fluid Coupled to an Elastic Membrane’’

An algorithm of Charles S. Peskin for simulation of incompressible flow of a fluid coupled to an elastic membrane, was implemented as a program for an asynchronous shared memory parallel computer, the ‘‘NYU-Ultracomputer’’. The algorithm is an iterative repetition of a cycle consisting of recomputing the position of the membrane and the velocity of the fluid for successive discrete times. Each iteration requires solving Navier-Stokes equations for the fluid, using an FFT and algorithms for solving linear equations and also includes an iterative minimization of a tension functional for recomputation of the new position of the membrane. All these phases are parallelized by spawning a number of independent tasks and waiting for all of them to terminate. The program includes subroutines REQUEST (INDEX) and SYNCH which provide coordination necessary for such a spawning. Timing analysis of simulated executions predicts high efficiency of this algorithm for parallel computers that include up to hundreds of processing elements in the 2-D case and up to thousands of processing elements in the 3-D case.

43. Ronald Bianchini and Ronald Bianchini, Jr., ‘‘Wireability of an Ultracomputer’’

We analyze the wireability of an Ultracomputer, a shared-memory MIMD parallel machine with thousands of processing elements, wired using an Omega connection network. We show a way of structuring hardware involving large numbers of processors into smaller subsets of processors and switches. A design with 4096 processors, based on present-day wiring technology and electrical power considerations, is presented. This design assumes a 2 by 2 switch on a chip is available, also a 1 megabit memory chip and a high speed microprocessor chip with 32 bit addressing.

44. Marc Snir, ‘‘On Parallel Searching’’

We investigate the complexity of searching by comparisons a table of n elements on a synchronous, shared memory parallel computer with p processors. We show that $\theta(\lg n)$ steps are required if concurrent accesses to the same memory cell are not allowed, whereas only $\theta(\lg n / \lg p)$ steps are required if simultaneous reads area allowed. We next show that these lower bounds are still valid even if only communication steps are counted, and even if polynomial comparisons are allowed.

45. B.D. Lubachevsky, ‘‘An Approach to Automating the Verification of Compact Parallel Coordination Programs’’

A class of parallel coordination programs for a shared memory asynchronous parallel processor is considered. They use the primitive operation Fetch and Add and differ in various respects from those which use basic P and V operations. The F&A is the basic primitive for the ‘‘NYU-Ultracomputer’’. A correctness proof for the considered programs must be done for arbitrary number N of processing elements since the Ultracomputer design includes thousands of PEs.

46. Malvin H. Kalos and Y-Q. Zhong, ‘‘Monte Carlo Transports on an Ultracomputer’’

This report summarizes the results of experiments in adapting transport Monte Carlo programs to an NYU Ultracomputer. Two fixed source (one in adjoint mode) and two eigenvalue problems were studied. In general, the required program is very straightforward: the shared memory MIMD architecture seems ideal for this application. Certain secondary questions related to efficiency loss owing to the straggling of histories are analyzed an practical remedies suggested and tested. The issue of repeatability of the Monte Carlo is solved with the device of having each process seed its successors. Finally, the probability that different pseudo-random sequences overlap in part is analyzed and recommendations are made for acceptable periods.

47. Herbert Bernstein and Max Goldstein, ‘‘RINSE – What Follows WASHCLOTH’’

This note describes a simulator for a parallel processor based on Gottlieb’s WASHCLOTH simulator. RINSE includes both the replace-add synchronization primatives of WASHCLOTH, and the full-empty synchronization primatives of the Denelcor HEP, to allow an exploration of the actual effect of choice of synchronization techniques. A linear equation solver was tested. In this case the degradation due to use of full-empty synchronization was found to occur with a number of PEs at which overall efficiency was also declining sharply.

48. Malvin H. Kalos, ‘‘The NYU Ultracomputer’’

The architecture of the NYU Ultracomputer is outlined. This is a proposed MIMD parallel machine comprising thousands of independently programmable fast processing elements connected to a large shared memory by a switching network. The latter has the geometry of an Omega-network, with switches each having enough memory and processing power to yield a bandwidth proportional to the number of processors. The network, which is pipelined, also supports the parallel execution of the ‘‘fetch-and-add’’ primitive important for process coordination. Questions relating to operating systems, programmability, and efficiency of applications codes are also discussed.

49. B.D. Lubachevsky, ‘‘An Approach to Automating the Verification of Compact Parallel Coordination Programs, II’’

This paper describes an algorithm for building reachability set descriptions for compact parallel programs. The notion of a controlled vector addition system (CVAS, which generalizes that of a vector addition system, is introduced. An algorithm which exhausts all the reachable states of a CVAS is described. It is proved that if the program to which the CVAS corresponds is normal, then termination of this algorithm is equivalent to the compactness of the program.

50. Uzi Vishkin, ‘‘On Choice of A Model of Parallel Computation’’

The problem of choosing a theoretical abstract model of parallel computation to be simulated by an exclusive-read, exclusive-write parallel RAM or a synchronous distributed machines is discussed. The principle of choosing the most permissible model of parallel computation as long as the cost of computational resources does not increase is applied.

We prove two theorems. The first (resp. second) theorem asserts that for every exclusive-read exclusive-write parallel RAM (resp. synchronous distributed machine) and every simulation of the concurrent-read exclusive-write parallel RAM model of computation into this machine, there exists a simulation of the Fetch-and-Add parallel RAM into the same machine that uses the same order of computational resources. It implies the choice of a Fetch-and-Add parallel RAM model of computation.

A promising methodological notion, called Execution Reducibilities, is being exercised in the proof of the main theorems.

51. Robert Tarjan and Uzi Vishkin, ‘‘0 (log n) and Optimal Parallel Bioconnectivity’’

In this paper we propose a new algorithm for finding the blocks (bioconnected components) of an undirected graph. A serial implementation runs in 0(n + m) time and space on a graph of n vertices and m edges. A parallel implementation runs in 0(log n) time and 0(n + m) time and 0(n + m) space using 0(n + m) processors on a concurrent-read, concurrent-write parallel RAM. An alternative implementation runs in $0(n^2/p)$ time and $0(n^2)$ space using an number $p \leq n^2/log^2 n$ of processors, on a concurrent-read, exclusive-write parallel RAM. The latter algorithm has optimal speed-up, assuming an adjacency matrix representation of the input.

52. W. Paul, Uzi Vishkin and H. Wagener, ‘‘Parallel Computation on 2-3 Trees’’

Our model of computation is a parallel computer with k synchronized processors $P_1, \ldots, P_k$ sharing a common random access storage, where simultaneous access to the same storage location by two or more processors is not allowed. Suppose a 2-3 tree T with n leaves is implemented in the storage, suppose $a_1, \ldots, a_k$ are data that may or may not be sorted in the leaves, and for all i, 1<i<k, processor P_i knows a_i. We show how to search for $a_i, \ldots, a_k$ in the tree T, insert these data into the tree, delete them from the tree, split the tree with respect to these elements and perform the union of k + 1 range-disjoint 2-3 trees in 0(log n + log k) steps.

53. Uzi Vishkin, ‘‘Synchronous Parallel Computation – A Survey’’

This survey is motivated by the fact that the tremendous potential power of microstructure technology can be realized only if we find effective parallel architectures and algorithms for utilizing large numbers of small but powerful processors. Many groups, including the very active NYU Ultracomputer group, are considering the pragmatic questions involved in the choice of effective parallel architectures. Theoretical studies, like that on which this presentation focuses, can buttress this pragmatic work in two ways: by finding parallel algorithms that use such machines effectively, and by defining abstract models of parallel computing thereby clarifying the ‘‘design space’’ within which the computer architect can make choices.

54. Uzi Vishkin and Avi Wigderson, ‘‘Dynamic Parallel Memories’’

Say that a parallel algorithm that uses p processors and N (> p) shared memory locations is given. The problem is simulating this algorithm by p processors and only p shared memory locations without increasing the running time by more that a constant factor is considered. A solution for a family of such parallel algorithms is given. The solution utilizes the idea of dynamically changing locations of the addresses of the algorithm throughout the simulation.

55. David Korn and Norman Rushfield, ‘‘WASHCLOTH Simulation of Three-Dimensional Weather Forecasting Codes’’

This note describes the method used, and the results obtained for simulating an existing weather forecast program of Rivas with the WASHCLOTH simulator. The WASHCLOTH simulator is used to investigate how FORTRAN programs would behave on an idealized MIMD machine (the paracomputer) which will be approximated by the NYU Ultracomputer. This memo sheds light on the problems encountered in taking existing codes and running them on the Ultracomputer.

56. Uzi Vishkin and Avi Wigderson, ‘‘Trade-Offs Between Depth and Width in Parallel Computation’’

A new technique for proving lower bounds for parallel computation is introduced. This technique enables us to obtain, for the first time, non-trivial tight lower bounds for shared-memory models of parallel computation that allow several processors to have simultaneous access to the same memory location. Specifically, we use a concurrent-read, concurrent-write model of parallel computation. It has *p* processors, each has access to a common memory of size *m* (also called communication width, or width in short). The input to the problem is located in an additional read-only portion of the common memory.

For a wide variety of problems (including parity, majority and summation) we show that the time complexity *T* (depth) and the communication width *m* are related by the trade-off curve $mT^2 = \Omega(n)$ (where *n* is the size of the input), regardless of the number of processors. Moreover, for every point on this curve with $m = 0(n/log^2 n)$ we give a matching upper bound with the optimal number of processors.

We extend our technique to prove $mT^3 = \Omega(n)$ trade-off for a class of ‘‘simpler’’ functions (including Boolean OR) on a weaker model that forbids simultaneous write access. We also state and give proof of a new result by Beame that achieves a tight lower bound for the OR in this model, namely $mT^2 = \Omega(n)$. These results improve the lower bound of Cook and Dwork when communication is limited.

57. Malcolm C. Harrison, ‘‘An Ultracomputer Switch Design Using Circuit and Packet Switching’’

In this note we show that and 8 by 8 switch, which will perform the function of 12 2 by 2 switches, can be built using 16 59-pin chips, 18 58-pin chips and 4 49-pin chips. This is better than 3 59-pin chips per 2 by 2 switch. If 296-pin chips are available, only 4 chips would be required for the 8 by 8 switch, less than 1/3 chip per 2 by 2 switch.

The 8 by 8 switch discussed is compatible with the proposed Ultracomputer physical design, and also with currently available chip packaging. It also appears to be competitive with pin counts up to several hundred, though with substantial increases in pin counts, 16 by 16 or larger switches may also be feasible.

58. Uzi Vishkin, ‘‘A Parallel Design Distributed Implementation (PDDI) General Purpose Computer’’

A scheme of an efficient general-purpose parallel computer is introduced. Its design space (i.e., the model for which parallel programs are written), is a permissive parallel RAM model of computation. The implementation space is presented as a scheme of a synchronous distributed machine which is not more involved that a sorting network followed by a merging network. An efficient translation from the design space into the implementation space is given. Suppose for some t and x there is a parallel algorithm in the design space which has depth (i.e., parallel time), 0(t/p) using p processors for all p<x. This translated to an algorithm in the implementation space with depth 0(t/s) for all $s<t/\ell$, where ℓ depends on the choice of the sorting and merging networks, s is the number of ‘‘powerful’’ processors used (processors not in the sorting or merging networks) and f(s,m) = 0(slog2s + mlogm), comparing favorably with alternative known solutions. Since many parallel algorithms are designed for a wide range of processors, our solution pays the fine for implementation where it hurts least.

59. Kurt Melhorn and Uzi Vishkin, ‘‘Granularity of Parallel Memories’’

Consider algorithms which are designed for shared memory models of parallel computation in which processors are allowed to have fairly unrestricted access patterns to the shared memory. General fast stimulations of such algorithms by parallel machines in which the shared memory is organized in modules where only one cell of each module can be accessed at a time are processed.

The paper provides a comprehensive study of the problem. The solution involves three stages:

Before a simulation, distribute randomly the memory addresses among the memory modules.

Keep several copies of each address and assign memory requests of processors to the ‘‘right’’ copies at any time.

Satisfy these assigned memory requests according to specifications of the parallel machine.

60. Allan Gottlieb, ‘‘Comparing the NYU Ultracomputer with other Large-Scale Parallel Processors’’

We describe the proposed NYU Ultracomputer, a shared memory MIMD parallel machine composed of thousands of autonomous processing elements. This machine uses an enhanced message switching network having the topology of an Ω-network to approximate the ideal of conflict-free access to a common memory and to implement efficiently a new fetch-and-add synchronization primitive. We outline the hardware required to construct such a system, consisting of 4096 processors, using 1990 technology and refer to other work indicating how the goal of a distributed operating system free from serial bottlenecks can be achieved by employing the fetch-and-add primitive. Finally, we compare the Ultracomputer project with other research in parallel processing.

61. Ilan Bar-On and Uzi Vishkin, ‘‘Optimal Parallel Generation of a Computation Tree Form’’

Given a general arithmetic expression, we find a computation binary tree representation in O(log n) time using n/log n processors.

62. Allan Gottlieb, ‘‘Avoiding Serial Bottlenecks in Ultraparallel MIMD Computers’’

Since, for any highly parallel architecture, the relative cost of a serial bottleneck rises linearly with the number of processors involved, users of such machines will be anxious to avoid critical, and hence necessarily serial, code sections even if the critical sections are sufficiently short not to cause performance penalties on today’s machines. The NYU Ultracomputer is a Multiple Instruction stream Multiple Data stream (MIMD) shared memory parallel computer designed to greatly reduce the need for critical sections. The present report describes HW and software considerations that arise and compares the Ultracomputer with other proposed parallel processors.

63. Marvin Bishop, ‘‘The Ultracomputer as a Vehicle for Polymer Situations’’

In this note, key concepts of parallelism relating to both hardware and software are reviewed. The NYU Ultracomputer design is treated. One particular application area – the simulation of polymer systems via the technique of molecular dynamics – is discussed.

64. Uzi Vishkin, ‘‘An Optimal Parallel Algorithm for Selection’’

A parallel algorithm for the problem of selecting the k-th smallest out of n elements is presented. It runs in 0(n/p) time using $p<n/(lognloglogn)$ processors on an exclusive-read exclusive-write parallel RAM.

65. Uzi Vishkin, ‘‘Lucid-Boxes vs. Black-Boxes’’

An explicit understanding of the opportunity for constructing new algorithms out of existing (or supposedly existing) algorithms is presented.

Say that $B, A_1, A_2, \ldots$ are problems. We present an abstract setting that provides for the effective use of algorithms for problems $A_1, A_2, \ldots$ for the design of efficient algorithms for problem B. A notion of ‘‘lucid-boxes’’ which is an extension of ‘‘black-boxes’’ is introduced for this purpose.

We exemplify the applicability of these lucid-box compositions (or reducibilities) for design and specification of efficient algorithms and when multi-parameter complexity optimization is required. Multi-parameter optimization is typical to parallel and distributed computation environments, where there is need to optimize simultaneously: time, size or local memories, communication load or many lines, etc.

66. Uzi Vishkin, ‘‘Randomized Speed-Ups in Parallel Computation’’

The following problem is considered: given a linked list of length n, compute the distance of each element of the linked list from the end of the list. The problem has two standard deterministic algorithms: a linear time serial algorithm, and an 0((n log n)/p + log n) time parallel algorithm using p processors. A known conjecture states that is is impossible to design an 0(log n) time deterministic parallel algorithm that uses only n/log n processors. We present three randomized parallel algorithms for the problem. One of these algorithms runs almost-surely in time of $0(n/p + \log n \log^* n)$ using p processors on an exclusive-read, exclusive-write parallel RAM.

67. Uzi Vishkin, ‘‘An Efficient Parallel Strong Orientation’’

The following problem is considered. *Input.* A connected bridgeless undirected graph G = (V,E,) $|V|$ = n, $|E|$ = m. *The strong orientation problem.* Assign directions to its edges to that the resulting diagraph is strongly connected. It is known that such strong orientation exists if the input graph is connected and bridgeless. We present an algorithm that runs in 0(log n) time using only n+m processors on a CRCW PRAM and an alternative implementation of the algorithm that runs in $0(n^2/p)$ time using $p \le n^2/log^2 n$ processors on an CREW PRAM. This is optimal for dense graphs.

68. Paul Spirakis, ‘‘Queuing Delay Modeling for Multistage Interconnection Multiprocessor Networks’’

We develop here a method of modeling the queuing delays in the discrete time, multistage, banyan networks. We proceed from modeling the queuing delays of a k by k switch of the first stage of the network to deriving the queue length distribution of the switch and then we analyze the more difficult cases of the internal network stages and the case of finite buffers. We extend here the analysis for queuing delays, given by Kruskal and Snir. Our results are in good agreement with simulation results. For example, our results can be applied to queuing delay modeling for the Ultracomputer network.

69. Jack Schwartz, ‘‘A Taxonomic Table of Parallel Computers Based on 55 Designs’’

A number of parallel computer designs are placed into a taxonomic table with three major classifications: Coarse Grained designs, Finegrained Designs, and Micro-Overlapped Serial Processor Designs. The first two classifications are further subdivided.

70. Seth Abraham, Allan Gottlieb, and Clyde P. Kruskal, ‘‘Simulating Shared-Memory Parallel Computers’’

The design and construction of parallel computers is expensive and time-consuming. Instruction level simulators represent one class of tools that can be used to ease this process. When compared to small-scale prototypes, instruction level simulators offer the potential for superior instrumentation. This paper presents a simulation technique for shared-memory parallel computers, and examines the use of this technique for two very different shared-memory parallel machines.

71. Malcolm C. Harrison, ‘‘Synchronous Combining of Fetch-and-Add Operations’’

An alternative architecture for combining F&A operations is proposed. The idea is to use two networks to connect the PEs and the memories. The first is the standard omega network, which handles all loads, all stores, and some F&As, but without combinings. The second is a smaller network which handles only F&As, and provides high-speed combining. We discuss here the organization of the second network. The origin of this approach is the observation that network requirements for F&A instructions which require combining are quite different from loads, stores, and other F&As. In particular, if there is a lot of combining, only a relatively small number of addresses are involved. Furthermore, if there are only as many addresses as stages in the network, they can be distributed synchronously in the network rather than randomly.

72. Robert A. Hummel, ‘‘Image Processing on the NYU Ultracomputer’’

Many intermediate and higher level vision tasks require long range communication of data within an image. As examples, we consider the connected component labeling problem, and the problem of finding convex hulls of planar regions in a pixel array. While these problems do not admit fast parallel implementations of a locally connected grid array of processors, on a shared memory parallel architecture, such as the NYU Ultracomputer, efficient algorithms can be easily devised. We sketch some of the issues in implementing parallel algorithms for these problems on an Ultracomputer.

73. Mikhail Atallah and Uzi Vishkin, ‘‘Finding Euler Tours in Parallel’’

The problem of finding Euler tours in directed and undirected Euler graphs is considered. We give 0(log $|V|$) time algorithms using a linear number of processors on a concurrent-read, concurrent-write parallel RAM.

74. Uzi Vishkin, ‘‘Optimal Parallel Pattern Matching In Strings’’

Given a text length n and a pattern, we present a parallel linear algorithm for finding all occurrences of the pattern in the text. The algorithm runs in 0(n/p) time using any number of p<n/log n processors on a concurrent-read, concurrent-write parallel random-access-machine.

75. Arzit Yannei, ‘‘PLISP Users’ Manual’’

This report describes an experimental implementation of PLISP, an parallel version of LISP. The implementation is written in YLISP, a dialect of LISP, which can simulate multiple processors on the VAX. Communication between parallel processors is via common structures and message queues.

76. Jack Schwartz, ‘‘Design Alternatives for Ultraperformance Parallel Computers’’

This article attempts to clarify the technical options available to parallel supercomputer development by listing various principal technical alternatives which characterize the design of such machines. Most of the parallel architectures that have been proposed occupy easily definable positions in the ‘‘design space’’ spanned by these ‘‘dimensions’’. Various machines typifying some of the many possibilities belonging to this space will be noted.

77. Allan Gottlieb, ‘‘A Remark on a Variable Length Shift Register’’

We note that the systolic queue of Guibas and Liang can be used to implement a varying length shift register and is also useful for shared memory multiprocessors.

78. UCN 78 HAS BEEN SUPERSEDED BY UCN 100.

79. Allan Gottlieb, ‘‘A Brief Update on the NYU Ultracomputer’’

The current status and near term future plans of the NYU Ultracomputer project are briefly presented.

80. Paul Spirakis, ‘‘Fast Probabilistic Techniques for Dynamic Parallel Addition, Parallel Counting, and the Processor Identification Program’’

We consider here three problems in synchronous parallel computation, with the common characteristic that the shared memory available to the processors is restricted. The first is the problem of *dynamic parallel addition*: n processors are given each keeping a number $x_{i,}\ i = 1,\ldots,n$. We know that only $m < n$ of the x_i ’s are nonzero but we don’t know in advance which processors have the nonzero numbers. We show how to compute the sum of the x_i’s in O(log m) *expected* parallel time and O(m) shared memory in the concurrent read - concurrent write model of parallel computation, through a probabilistic algorithm. We then consider the related problem of *processor identification*: n processors are given, each keeping either a 0 or a 1. We want each processor at the end to know which are the processors with the 1’s. The shared memory available is very small (say 1 location), and can store only O(n) size numbers. To solve this problem O(n) time is required in a parallel read-exclusive write (or even concurrent write) PRAM, even if O(n) shared memory is available.

We show how to do it in O(n/logn) parallel time, if we use a weak version of a (more powerful) model of parallel computation, the so-called *paracomputer*. Our algorithm is again probabilistic. For this algorithm, we use combinatorial ideas developed by Erdos and Renyi [ER,63] . Finally, we show how to do *approximate parallel counting* of O(n) units in O(1) parallel time, by using O(n/logn) shared memory, and an n-processor concurrent read - concurrent write machine.

81. Edith Schonberg and Edmond Schonberg, ‘‘Highly Parallel Ada – Ada on an Ultracomputer’’

We examine the suitability of Ada to a specific multiprocessor architecture, the NYU Ultracomputer. The Ultracomputer is an MIMD machine with shared memory, and a special hardware primitive, the *fetch-and-add* operation, with which non-blocking versions of well-known synchronization primitives can be implemented. We show that fetch-and-add can be used to implement some common Ada tasking idioms, and that well-known algorithms for MIMD machines find a natural description in Ada. Furthermore, Ada solutions implemented with fetch-and-add have fewer and smaller critical sections than previously presented versions. The fetch-and-add primitive also provides an efficient solution to the problem of initializing a number of identical tasks, and to the potentially expensive implementation of termination for large numbers of tasks. We conclude that the Ada tasking model ‘‘fits’’ well an MIMD architecture such as the Ultracomputer.

82. UCN 82 IS OUT OF PRINT.

83. Malvin H. Kalos, ‘‘Monte Carlo Methods and the Computers of the Future’’

A brief characterization is given for many Monte Carlo algorithms with respect to dimensionality, granularity, regularity of logical flow, interdependence of random walks, and communication patterns. These determine the properties of architectures that will support them well. The ‘‘Metropolis’’ algorithm, as applied to the two-dimensional Ising model, is discussed in some detail, along with issues related to optimality of the algorithm and of suitable architectures. Transport Monte Carlo, widely used to calculate the passage of particles and radiation through matter, is discussed more briefly. Vectorization is difficult and tends to degrade the desirable features of Monte Carlo; MIMD parallelization is rather easy. It is argued that for certain realistic large problems, shared memory architectures will be essential. A survey of the hardware and software properties of the proposed NYU Ultracomputer shows it to be very promising as a general purpose machine for all Monte Carlo applications.

84. UCN 84 IS OUT OF PRINT.

85. Susan Dickey, Richard Kenner, Marc Snir, and Jon Solworth, ‘‘A VLSI Combining Network for the NYU Ultracomputer’’

The NYU Ultracomputer architecture, a shared memory MIMD parallel machine composed of thousands of processing elements, requires a high-performance interconnection network to approximate the ideal behavior of Schwartz’s paracomputer model and to implement the fetch-and-add synchronization primitive efficiently. In this paper we describe the routing scheme and protocols for an enhanced message switching network and a VLSI implementation of a network node. Systolic queues at each node combine accesses to the same memory location by different processing elements allowing them to be performed in the same memory cycle. We have had encouraging results to date towards the goal of attaining network latency comparable with memory access times.

86. Richard Cole and Alan Siegel, ‘‘Optimal VLSI Circuits for Sorting’’

This work describes a large number of constructions for sorting N numbers in the range $[0, M]$ for the standard VLSI bit model. Among other results, we attain:

• VLSI sorter constructions that are within a constant factor of optimal size for almost all number ranges M (including $M = N$), and running times T.

• A fundamentally new merging network for sorting numbers in a bit model.

• New organizational approaches for optimal tuning of merging networks and the proper management of data flow.

87. Robert Hummel and Alan Rojer, ‘‘Connected Component Labeling in Image Processing with MIMD Architectures’’

Many higher level image processing algorithms suggest dynamic allocation of multiple processors to image processing subtasks. However, most parallel algorithms in image processing assume an SIMD architecture with processors statically assigned to one or a group of image pixels. For many vision tasks, fixed assignments of processors will inevitably lead to low parallelism. Instead, it seems natural to have an intermediate number of more powerful independent processors, each with access to the shared data, to perform tasks which tend to be more global in nature. In converting SIMD algorithms into efficient code for MIMD machines with relatively fewer processors, it can frequently be determined when processors are active and when they are dormant. By analyzing the dynamic flow of subtasks, we can attempt to assign processors to the portions of the image where non-null tasks would be accomplished in a more processor-rich environment.

We study, as an example, the Shiloach/Vishkin connected component algorithm, which is an O(logN) SIMD algorithm which requires roughly 4N processors, where N is the number of pixels. Each pixel points to another pixel in its connected component, forming a pointer graph which is always organized as a forest. Pointers are updated by a ‘‘leapfrogging’’ step and by a step which ‘‘hooks’’ trees together. We show how the algorithm may be converted to an MIMD algorithm, and consider issues of task flow and synchronization. We present code for an implementation of the MIMD version of the algorithm, and consider operating times on typical images. We show that similar performance can be using the MIMD approach as with the more standard mesh connected local iterative connected component method.

88. B.D. Lubachevsky and K.G. Ramakrishn, ‘‘Process-Level, Time-Driven, Simulation of a Computer Network on a Parallel Shared-Memory Processor’’

This paper reports on a parallel, time-driven simulation of a computer network designed to run on a shared-memory MIMD multiprocessor. The NYU Ultracomputer is chosen as the model of a shared-memory MIMD machine. Various results on efficiency and speedup are reported. It may be inferred from the results presented in the report, that a time-driven simulation, though inefficient in the uniprocessor context, can prove to be a useful tool in the parallel context. A major part of this paper is devoted to algorithms for coordination and synchronization of processors in the ultracomputer. A dynamic task manager for scheduling tasks is also discussed. Actual FORTRAN codes are presented for the synchronization and scheduling of processors. A brief overview of a new table-driven approach for simulating computer networks at process-level, is also presented. This method has the potential for enormous savings in development time of simulation programs. After a discussion of results on speedup and efficiency of our parallel simulator, a simple theoretical model is developed to predict the efficiency for larger computer networks. This method takes into account the synchronization overhead and concurrency of events.

89. Richard Cole and Uzi Vishkin, ‘‘Deterministic Coin Tossing With Applications to Optimal Parallel List Ranking’’

The following problem is considered: given a linked list of length n, compute the distance from each element of the linked list to the end of the list. The problem has two standard deterministic algorithms: a linear time serial algorithm, and an $O(\log n)$ time parallel algorithm using n processors. We present new deterministic parallel algorithms for the problem. Our strongest results are:

1. $O(\log n \log^{(k)} n)$ time using $n/(\log n \log^{(k)} n)$ processors, for any fixed positive integer k, where $\log^{(k)}$ is the k-th iterate of the log function. This algorithm achieves optimal speed-up.

2. $O(\log n \log^{*} n)$ time using $n/\log n$ processors. Since $\log^{*} n$ grows extremely slowly as a function of n this algorithm achieves optimal speed-up for all practical purposes.

3. $O(\log n)$ time using $n \log^{(k)} n/\log n$ processors, for any fixed positive integer k.

The algorithms apply a novel "random-like" deterministic technique. This technique provides for a fast and efficient breaking of a symmetric situation in parallel.

90. Richard Cole and Alan Siegel, ‘‘Lower Bounds on Communication Complexity in VLSI’’

We analyze a family of problems governing the VLSI complexity of broadcasting B bits of information in time T ($\leq B$) to each of N processors located along the perimeter of a two dimensional broadcast network. Among other results, we show:

• $AT^2 = \Theta(N B^2 \frac{\log N}{\log(2T/\log N)})$ for such network, when the output ports are unit distance apart. This bound seems to be the first AT^2 tradeoff which is not based on the number of bits that must cross a cut or boundary of some subcircuit.

• The above AT^2 bound scales linearly in the distance between adjacent ports, for ports distributed along the circuit perimeter with an approximately uniform spacing.

• Any vertex degree 3 graph, which has N vertices and breadth first search depth T, contains a complete binary tree of depth $\Omega(\frac{\log N}{\log(2T/\log N)})$.

These problems are of importance in the design and analysis of optimal VLSI circuits, such as sorters that have their input ports located along the circuit perimeter. Moreover, some of the proof techniques are of interest in their own right.

91. UCN 91 HAS BEEN SUPERSEDED BY UCN 100.

92. UCN 92 HAS BEEN SUPERSEDED BY UCN 100.

93. Susan Dickey, Richard Kenner and Marc Snir, ‘‘An Implementation of a Combining Network for the NYU Ultracomputer’’

The NYU Ultracomputer architecture, a shared memory MIMD design composed of thousands of processing elements, requires a high-performance interconnection network between its processors and memory modules. This network must be capable of merging concurrent requests to the same memory location from multiple processors. We present an implementation of such a network for a 32-processor configuration with emphasis on the mechanisms used to merge these concurrent requests.

94. UCN 94 HAS BEEN SUPERSEDED BY UCN 100. Susan Dickey, Allan Gottlieb, Richard Kenner ‘‘Using VLSI to Reduce Serialization and Memory Traffic in Shared’’ Memory Parallel Computers"

This paper appeared in Advanced Research in VLSI: Proceedings of the Fourth MIT Conference, Charles E. Leiserson, editor.

95. Dennis Shasha, ‘‘Query Processing in a Symmetric Parallel Environment’’

We consider a database machine consisting of n nodes connected by an O(n*processing speed) bandwidth network. Each node consists of a processor, a random access memory, and a slower but much larger memory such as a disk. In order to approach optimal (O(n)) speedup on this hardware architecture, we partition relations roughly evenly among the processors. We study the problem of optimizing multi-join queries assuming such a data distribution strategy. For us, optimization consists of minimizing the number of times we need to redistribute relations in the course of a query. The optimization problem is NP-complete for general queries but linear time for a subclass of tree queries.

96. Dennis Shasha and Marc Snir, ‘‘Efficient and Correct Execution of Parallel Programs that Share Memory’’

In this paper, we consider an optimization problem that arises in the execution of parallel programs on shared memory multiple-instruction stream multiple-data stream (MIMD) computers. A program on such a machine consists of many *program segments* each executed sequentially by a single processor. The processors have access to shared memory, and can execute standard memory access operations on this shared memory. This memory is distributed among many separate memory modules. A network connects processors to memory modules. Delays on this network are stochastic. Thus, operations issued by a processor to distinct memory modules may not be executed as memory requests on those modules in the order they were issued.

For performance reasons, we want to allow one operation to begin before a previous one in the same instruction. Our analysis gives a method for determining which operations in a stream may be issued concurrently without changing the semantics of the execution. We also consider code where blocks of operations have to be executed atomically. This introduces the necessity of locks. We use a conflict graph similar to that used to schedule transactions in distributed databases. Our graph incorporates the order on operations given by the program text, enabling us to do without locks even when database conflict graphs would suggest that locks are necessary.

97. Richard Cole, ‘‘An Optimal Selection Algorithm’’

We give an optimal parallel algorithm for selection on the EREW PRAM. It requires a linear number of operations and $O(\log n \log^* n)$ time.

98. Anne Greenbaum, ‘‘Synchronization Costs on Multiprocessors’’

Various types of processor synchronization are introduced and analyzed with regard to execution time and waiting time. It is shown that while barrier synchronization requires a large execution time on systems with many processors, other less restrictive forms of synchronization do not have this drawback. It is also shown that for most reasonable distributions of processor times, a relatively small amount of time is spent waiting at synchronization points. The conclusion is that if barriers can be replaced by the less restrictive synchronization forms, then, for problems with appropriate size granularity, the synchronization costs on most multiprocessors will be small.

99. Anne Greenbaum, ‘‘Solving Sparse Triangular Linear Systems Using FORTRAN with Parallel Extensions on the NYU Ultracomputer Prototype’’

A parallel method is presented for solving sparse triangular linear systems. Several implementations of this algorithm using FORTRAN with parallel extensions on the NYU Ultracomputer Prototype are described. Timing results are given, along with analysis of expected speedup. Possible techniques for improving performance are also discussed.

100. Allan Gottlieb, ‘‘An Overview of the NYU Ultracomputer Project’’

This report is an edited amalgam of papers I have coauthored with Susan Dickey, Jan Edler, Ralph Grishman, Richard Kenner, Clyde P. Kruskal, Jim Lipkis, Kevin P. McAuliffe, Larry Rudolph, Marc Snir, Pat Teller, and Jim Wilson. It provides information on the NYU Ultracomputer architecture, programming model, operating system, hardware and VLSI design, performance analysis, and simulation studies.

101. Gad M. Landau and Uzi Vishkin, ‘‘Efficient Parallel And Serial Approximate String Matching’’

Consider the string matching problem, where differences between characters of the pattern and characters of the text are allowed. Each difference is due to either a mismatch between a character of the text and a character of the pattern or a superfluous character in the text or a superfluous character in the pattern. Given a text of length n, a pattern of length m and an integer k, we present parallel and serial algorithms for finding all occurrences of the pattern in the text with at most k differences. The first part of the parallel algorithm consists of analysis of the pattern and takes $O(log\ m)$ time using m^2 processors. The rest of the algorithm consists of handling the text. The text handling part applies the following new approach. This part starts by obtaining a *concise* characterization of the text which is based solely on substrings of the pattern in $O(log\ m)$ time using $n/log\ m$ processors. Then the desired output is derived from this characterization together with the tables built in the first part in $O(k)$ time using n processors.

The serial algorithm follows also this new approach for handling the text. It runs in $O(kn)$ time for alphabet whose size is fixed. For general input the algorithm requires $O(n(k + \log m))$ time. In both cases the space requirement is $O(n)$.

102. Yael Maon, Baruch Schieber, and Uzi Vishkin, ''Parallel Ear Decomposition Search (EDS) And ST-Numbering in Graphs''

[LEC-67] linear time serial algorithm for testing planarity of graphs uses the linear time serial algorithm of [ET-76] for *st*-numbering. This *st*-numbering algorithm is based on depth-first search (DFS). A known conjecture states that DFS, which is a key technique in designing serial algorithms, is not amenable to poly-log time parallelism using ''around linearly'' (or even polynomially) many processors. The first contribution of this paper is a general method for searching efficiently in parallel undirected graphs, called *ear-decomposition search* (EDS).

The second contribution demonstrates the applicability of this search method. We present an efficient parallel algorithm for *st*-numbering in a biconnected graph. The algorithm runs in logarithmic time using a linear number of processors on a concurrent-read concurrent-write (CRCW) PRAM. An efficient parallel algorithm for the problem did not exist before. The problem was not even known to be in NC.

103. Yossi Azar and Uzi Vishkin, ''Tight Comparison Bounds On The Complexity of Parallel Sorting''

The problem of sorting n elements using p processors in a parallel comparison model is considered. Lower and upper bounds which imply that for $p \geq n$, the time complexity of this problem is $\Theta(\frac{logn}{log(1+p/n)})$ are presented. This complements [AKS-83] in settling the problem since the AKS sorting network established that for $p \leq n$ the time complexity is $\Theta(\frac{n\log n}{p})$. To prove the lower bounds we show that to achieve $k \leq \log n$ parallel time, we need $\Omega(n^{1+1/k})$ processors.

104. Malcolm C. Harrison, ''The Add-and-Lambda Operation: An Extension of F&A''

This note describes a proposed extension to the architecture of shared memory multiprocessors with combining fetch-and-add operations, such as the NYU Ultracomputer and the IBM RPn. The extension involves addition of a small amount of hardware between the network and the memory, which permits the efficient implementation of a number of parallel operations. Examples are given.

105. Clyde P. Kruskal, Larry Rudolph, and Marc Snir, ''Efficient Synchronization on Multiprocessors with Shared Memory''

A new formalism is given for read-modify-write (RMW) synchronization operations. This formalism is used to extend the memory reference combining mechanism, introduced in the NYU Ultracomputer, to arbitrary RMW operations. A formal correctness proof of this combining mechanism is given. General requirements for the practicality of combining are discussed. Combining is shown to be practical for many useful memory access operations. This includes memory updates of the form *mem_val := mem_val op val*, where *op* need not be associative, and a variety of synchronization primitives. The computation involved is shown to be closely related to parallel prefix evaluation.

106. Clyde P. Kruskal and Marc Snir, ''A Unified Theory of Interconnection Network Structure''

The relationship between the topology of interconnection networks and their functional properties is examined. Graph theoretical characterizations are derived for delta networks, which have a simple routing scheme, and for bidelta networks, which have the delta property in both directions. Delta networks are shown to have a recursive structure. Bidelta networks are shown to have a unique topology. The definition of bidelta network is used to derive in a uniform manner the labeling schemes that define the omega networks, indirect binary cube networks, flip networks, baseline networks, modified data manipulators, and two new networks; these schemes are generalized to arbitrary radices.

The labeling schemes are used to characterize networks with simple routing. In another paper, we characterize the networks with optimal performance/cost ratio. Only the multistage shuffle-exchange networks have both optimal performance/cost ratio and simple routing. This helps explain why few fundamentally different geometries have been proposed.

107. Susan Flynn, Edith Schonberg, and Edmond Schonberg, ''The Efficient Termination of Ada Tasks in a Distributed Environment''

An efficient implementation of Ada task termination for shared memory MIMD architectures is presented. The solution is highly parallel and distributed; termination is not synchronized by a centralized supervisor, and there are no critical sections. The handling of block and subprogram termination that depend on nested tasks is also included. This implementation can be easily adapted to message based systems.

108. Richard Cole and Uzi Vishkin, ''The Accelerated Centroid Decomposition Technique for Optimal Parallel Tree Evaluation In Logarithmic Time''

A new general parallel algorithmic technique for computations on trees is presented. The new technique performs a reduction of the tree expression evaluation problem to list ranking; then, the list ranking provides a schedule for evaluating the tree operations. The technique needs logarithmic time using an optimal number of processors and has applications to other tree problems.
This new technique enables us to systematically order four basic ideas and techniques for parallel algorithms on tree: (1) The list ranking problem. (2) The Euler tour technique on trees. (3) The centroid decomposition technique. (4) The new *accelerated centroid decomposition (ACD)* technique.

109. Richard Cole and Colm O'Dunlaing, ''Notes on the AKS Sorting Network''

The Ajtai-Komlos-Szemeredi sorting network, as improved and simplified by Michael S. Paterson, is described.

110. Richard Cole and Uzi Vishkin, ''Approximate Parallel Scheduling: Part I: The Basic Technique with Applications to Optimal Parallel List Ranking in Logarithmic Time''

We define a novel scheduling problem; it is solved in parallel by repeated, rapid, approximate reschedulings. This leads to the first optimal logarithmic time PRAM algorithm for list ranking. Companion papers show how to apply these results to obtain improved PRAM upper bounds for a variety of problems on graphs, including: connectivity, biconnectivity, minimum spanning tree, Euler tour and st-numbering, and a number of problems on trees.

111. Gad M. Landau, Baruch Schieber and Uzi Vishkin, ''Parallel Construction of a Suffix Tree''

Weiner's [We-73] suffix tree is known to be a powerful tool for string manipulations. We present a parallel algorithm for constructing a suffix tree. The algorithm runs in 0(logn) time and uses n processors. We also present applications for designing efficient parallel algorithms for several string problems.

112. Isaac Dimitrovsky, ‘‘A Group Lock Algorithm, With Applications’’

In this note I introduce the concept of a group lock. This is a generalization of P and V that can be used in writing asynchronous parallel algorithms. I then present and algorithm for group locking that has a nice time complexity property. This algorithm is used to define several fast and simple parallel algorithms. Among these are algorithms for a parallel queue, a parallel stack, atomic read/writes on multiword records, a parallel heap, and a parallel dictionary.

113. Ora E. Percus, ‘‘A Note on Random Number Generator of Chung et. al.’’

We derive the probability distribution $P\{X_n = x\}$ for a random walk satisfying $X_0 = 0$; $P\{X_{n+1} = (2X_n + 1) \bmod p\} = P\{X_{n+1} = (2X_n - 1) \bmod p\} = \frac{1}{2}$ for $n \geq 0$ and p odd integer.

114. Ora E. Percus and Malvin H. Kalos, ‘‘Random Number Generators for Ultracomputers’’

We discuss and analyze issues related to the design of pseudorandom number generators (prn’s) for MIMD (multiple instruction stream/multiple data stream) parallel processors, which are very well suited for Monte Carlo calculations. We are concerned to ensure reproducibility of runs, to provide very long sequences, and to assure an adequate degree of independence of the parallel streams. We consider the class of linear congruential generators

$$x_{n+1,i} \equiv a x_{n,i} + b_i \mod m$$

and analyze the effect that different choices of b_i have on the correlation properties between such streams. We derive a spectral test ν_t for t parallel linear congruential generators, a modification of Knuth’s Algorithm **S**. From this, we prove a good lower bound for $\nu_2^* = \min_{\text{all pairs}(i,j)} \nu_2(i,j)$ for certain choices of b_i’s. The set of the largest r primes p_i, $i = 1, \ldots, r$ satisfying $p_i < \sqrt{m/2}$, where m is the period length of each generator gives a lower bound $O(m^{1/2})$ to the correlation between a pair of corresponding elements in any two streams. An alternative choice, $b_i = d^i \bmod m$ for $d = m^{1/2} + 1$ gives a bound $O(m^{1/2}/(t-1))$ which will be satisfactory for small numbers of streams. Finally, we construct a spectral test for correlations between $x_{n,i}$ and $x_{n+k,i+l}$, but derive no analytic prescriptions from it.

115. Richard Cole, ‘‘Parallel Merge Sort’’

We give a parallel implementation of merge sort on a CREW PRAM that uses n processors and $O(\log n)$ time; the constant in the running time is small. We also give a more complex version of the algorithm for the EREW PRAM; it also uses n processors and $O(\log n)$ time. The constant in the running time is still moderate, though not as small.

116. Malcolm C. Harrison, Nikolaos Markantonatos and Georgios Papadopoulos, ‘‘UltraProlog’’

UltraProlog is a dialect of Prolog designed for shared memory parallel computers such as the NYU Ultracomputer and the IBM RPx, and is being implemented on the NYU Ultracomputer. UltraProlog is similar in spirit to Concurrent Prolog, Parlog and Guarded Horn Clauses. This report defines the language, and describes the implementation approach.

117. Richard Cole and Uzi Vishkin, ‘‘Faster Optimal Parallel Prefix Sums and List Ranking’’

We present a parallel algorithm for the prefix sums problem which runs in time $O(\log n / \log\log n)$ using $n \log\log n / \log n$ processors (optimal speed up). This algorithm leads to a parallel list ranking algorithm which runs in $O(\log n)$ time using $n/\log n$ processors (optimal speed up).

118. Baruch Schieber and Uzi Vishkin, ''On Finding Lowest Common Ancestors: Simplification and Parallelization''

We consider the following problem. Suppose a rooted tree T is available for preprocessing. Answer on-line queries requesting the lowest common ancestor for any pair of vertices in T. We present a linear time and space preprocessing algorithm which enables us to answer each query in $O(1)$ time, as in Harel and Tarjan [HT-84]. Our algorithm has the advantage of being simple and easily parallelizable. The resulting parallel preprocessing algorithm runs in logarithmic time using an optimal number of processors on an EREW PRAM. Each query is then answered in $O(1)$ time using a single processor.

119. Richard Cole and Uzi Vishkin, ''Approximate Parallel Scheduling. Part II: Applications to Optimal Parallel Graph Algorithms in Logarithmic Time''

Part I of this paper presented a novel technique for approximate parallel scheduling and a new logarithmic time optimal parallel algorithm for the list ranking problem. In this part, we give a new logarithmic time parallel (PRAM) algorithm for computing the connected components of undirected graphs which uses this scheduling technique. The connectivity algorithm is optimal unless $m = o(n \log^* n)$††, in graphs of n vertices and m edges. Using known results, this new algorithms imply logarithmic time optimal parallel algorithms for a number of other graph problems, including: biconnectivity, Euler tours, strong orientation and st-numbering. Another contribution of the present paper is a parallel union/find algorithm.

To the best of our knowledge, this paper presents for the first time optimal, logarithmic time, parallel algorithms for problems on graphs.

120. Vladimir Fleygakker, ''On Some Parallel Sieve of Prime Number Generation''

This short report addresses the question of using The NYU Ultracomputer for the generation of prime numbers.

121. Ora E. Percus and Jerome K. Percus, ''Long Range Correlations in Linear Congruential Generators''

Every linear congruential random number generator that uses a power of two as modulus introduces correlations among elements separated sequentially by powers of two. In particular, we have the following Theorem:

Let

$$x_{n+1} = ax_n + b \mod 2^{\beta}$$

where

$$a \equiv 1 \mod 2^{\alpha} \quad \text{and} \quad a \neq \mod 2^{\alpha+1} \quad \text{and} \quad \beta > \alpha > 1.$$

Then for every k the linear relation

$$\sum_{j=0}^{p} c_j x^k_{n+j2} \equiv 0 \mod 2^{\beta}$$

holds, where $c_j = (-1)^j \begin{bmatrix} p \\ j \end{bmatrix}$ p is the smallest integer $\geq \begin{matrix} (\beta+\alpha)/(k+\alpha) & \text{if } b \text{ is odd} \\ \beta/k+\alpha & \text{if } b = 0 \end{matrix}$

122. Kaizhong Zhang and Dennis Shasha, ''On the Editing Distance Between Trees and Related Problems''

Since a tree can represent a scene description, a grammar parse, a structural description, and many other phenomena, comparing trees is a way to compare scenes, parses and so on. We consider the distance

between two trees to be the (weighted) number of edit operations (insert, delete, and modify) to transform one tree to another. Then, we consider the following kinds of questions:
1. What is the distance between two trees?
2. What is the minimum distance between T_1 and T_2 with an arbitrary subtree removed? What if zero or more subtrees can be removed from T_2? A specialization of these algorithms solves the problem of approximate tree matching.
3. Given k, are T_1 and T_2 within distance k of one another and if so what is their distance?

We present a postorder dynamic programming algorithm to solve question 1 in sequential time $O(|T_1| \times |T_2| \times depth(T_1) \times depth(T_2))$ compared with $O(|T_1| \times |T_2| \times (depth(T_1))^2 \times (depth(T_2))^2)$ for the best previous algorithm due to Tai. Further, our algorithm can be parallelized to give time $O(|T_1| + |T_2|)$. Our approach extends to answer question 2 giving an algorithm of the same complexity. For question 3, a variant of our distance algorithm yields an $O(k^2 \times \min(|T_1|, |T_2|) \times \min(depth(T_1), depth(T_2)))$ algorithm.

123. Robert A. Hummel and Kiazhong Zhang, ‘‘Dynamic Processor Allocation for Parallel Algorithms in Image Processing’’

Because image processing is numerically intensive, there has been much interest in parallel processing for image analysis applications. While much of low-level vision can be attacked by SIMD mesh-connected architectures, intermediate and high-level vision applications might be able to make effective use of MIMD and distributed architectures. We have taken a standard parallel connected components algorithm, and applied it to image segmentation using an MIMD architecture. The resulting version of the Shiloach/Vishkin algorithm runs on the prototype NYU Ultracomputer. We will describe the implementation and the results of some experiments. We take note of the lesson learned from this implementation: that processor power should be focused dynamically to those portions of the image requiring greatest attention. We then consider the implications of this lesson to other image processing tasks.

124. Dennis Shasha, Vladimir Lanin and Jeanette Schmidt, ‘‘An Analytical Model for the Performance of Concurrent B Tree Algorithms’’

A dictionary is an abstract data type supporting the actions search, insert, and delete. Search structures are data structure used to implement a dictionary, e.g. B trees, hash structures, grid files, etc. Many concurrent algorithms for search structures have been proposed, but relatively little work has addressed their performance. This paper proposes an analytic performance model for a range of concurrent algorithms on B+ trees. It uses this model to propose a high performance, simple-to-program variant of lock-coupling algorithms.

The model actually consists of two mutually exclusive models: a ‘‘constant flow’’ model in which the throughput is assumed to remain at its average value; and a ‘‘bursty flow’’ model in which the throughput is predicted to oscillate with a certain frequency and amplitude based on the number of processes, the height of the tree, and the size of the nodes. We find that the data points from simulation fall between these two predictive models, tending towards the bursty flow one as the number of processors increases. Inasmuch as the analytical model characterizes B+ trees in terms of fanout, height, probability of restructuring, and node size, it does not depend on the details of B+ trees and so can be extended, at least in spirit, to other structures.

125. Susan Dickey, Allan Gottlieb, Richard Kenner, and Yue-Sheng Liu, ‘‘Designing VLSI Network Nodes To Reduce Memory Traffic in a Shared Memory Parallel Computer’’

Serialization of memory access can be a critical bottleneck in shared memory parallel computers. The NYU Ultracomputer, a large-scale MIMD (multiple instruction stream, multiple data stream) shared memory architecture, may be viewed as a column of processors and a column of memory modules connected by a rectangular network of enhanced 2 x 2 buffered crossbars. These VLSI nodes enable the network to combine multiple requests directed at the same memory location. Such requests include a new coordination primitive, fetch-and-add, which permits task coordination to be achieved in a highly parallel manner. Processing within the network is used to reduce serialization at the memory modules.

To avoid large network latency, the VLSI network nodes must be high-performance components. Design tradeoffs between architectural features, asympotic performance requirements, cycle time, and packaging limitations are complex. This report sketches the Ultracomputer architecture and discusses the issues involved in the design of the VLSI enhanced buffered crossbars which are the key element in reducing serialization.

126. Anne Greenbaum, Congming Li, and Han Zheng Chao, ‘‘Comparison of Linear System Solvers Applied to Diffusion-Type Finite Element Equations’’

Various iterative methods for solving the linear systems associated with finite element approximations to self-adjoint elliptic differential operators are compared based on their performance on serial and parallel machines. The methods studied are all preconditioned conjugate gradient methods, differing only in the choice of preconditioner. The preconditioners considered arise from diagonal scaling, incomplete Cholesky decomposition, hierarchical basis functions, and a Neumann-Dirichlet domain decomposition technique. The hierarchical basis function idea is shown to be especially effective on both serial and parallel architectures.

127. Mikhail J. Atallah, Richard Cole, and Michael T. Goodrich, ‘‘Cascading Divide-and-Conquer: A Technique for Designing Parallel Algorithms’’

We present techniques for parallel divide-and-conquer, resulting in improved parallel algorithms for a number of problems. The problems for which we give improved algorithms include intersection detection, trapezoidal decomposition, and planar point location. We also give efficient parallel algorithms for fractional cascading, 3-dimensional maxima, 2-set dominance counting, and visibility from a point. All of our algorithms run in $O(\log n)$ time with either a linear or sub-linear number of processors in the CREW PRAM model.

128. Richard Cole and Ofer Zajicek, ‘‘On Optimal Parallel Algorithm for Building A Data Structure for Planar Point Location’’

Using a generalization of the Approximate Parallel Scheduling of Cole and Vishkin [CoV87a], we give an optimal parallel algorithm for generating a linear size data structure for planar point location that supports an $O(\log n)$ query time. Our algorithm runs on a EREW PRAM in time $O(\log n)$ using a $n/\log n$ processors

129. Patricia J. Teller, Richard Kenner, and Marc Snir, ‘‘TLB Consistency on Highly-Parallel Shared-Memory Multiprocessors’’

Multiprocessors that store the same shared data in different private caches must ensure these caches have consistent copies. Almost all known solutions to this *cache consistency* problem are only suitable for architectures with a few tens of processors (PEs). Efficient solutions to the *TLB (translation lookaside buffer) consistency problem*, a special case of the cache consistency problem, can be found for *highly-parallel, shared-memory multiprocessors* (HPSMMs) with many hundreds of PEs for the following reasons: the number of references to address translation information per modification is very large; the cache for storing translation information can be present anywhere on the path from the PEs to memory; when the memory mapping needs to be modified, one can often select which translation information to change; and obsolete mapping information can be used until permanent changes must be made. We present three general methods that exploit these features and can be used on HPSMMs to maintain TLB consistency. Tradeoffs are discussed and are related to overall system performance. Some interesting issues inherent to the TLB consistency problem and the support of a demand-paged virtual memory system on a HPSMM are also presented. Our methods demonstrate the use of alternative implementations of the classical readers-writers algorithm.

130. Marsha Berger, ‘‘Adaptive Mesh Refinement for Parallel Processors’’

131. Maksymilian Dryja and Olof Widlund, ‘‘An Additive Variant of the Schwarz Alternating Method for the Cade of Many Subregions’’

In recent years, there has been a revival of interest in the Schwarz alternating method. Other domain decomposition algorithms, in particular the so called iterative substructuring methods, have also been developed to solve elliptic finite element problems. In this paper, we present an additive variant of the Schwarz method for elliptic problems, which shows great promise for parallel computers. By using techniques previously developed for iterative substructuring methods, we are able to show that this methods converges quite rapidly even when the region is divided into many subregions. We note that, as is the case with other fast iterative methods for many subregions, a mechanism for the global transportation of information is necessary in order to obtain fast convergence.

132. Yue-Sheng Liu, ‘‘Delta Network Performance and ‘‘Hot Spot’’ Traffic’’

In this paper, we review previous work on both buffered and unbuffered delta network performance analysis. We then analyze the delta network performance under ‘‘hot spot’’ traffic and give a solution to the single ‘‘hot spot’’.

133. Theodore Johnson, ‘‘Modifying Two-Phase Locking to Improve Performance’’

Studies of concurrency control performance suggest that Two-Phase Locking (2PL) suffers from data-contention thrashing because transaction blocking is uncontrolled, allowing blocked transactions to block other transactions. Our algorithm improves the performance of 2PL in certain situations by aborting blocked transactions that block other transactions. This paper presents an analysis that validates this claim. The new algorithm gives better performance if aborting transactions is not too expensive.

134. Yehuda Afek, Gad M. Landau, Baruch Schieber, and Moti Yung, ‘‘The Power of Multimedia: Combining Point-to-Point and Multiaccess Networks’’

In this paper we introduce a new network model called a *multimedia network*. It combines the point-to-point message passing network and the multiaccess channel. To benefit from the combination we design algorithms which consist of two stages: a local stage which utilizes the parallelism of the point-to-point network and a global stage which utilizes the broadcast capability of the multiaccess channel. As a reasonable approach, one wishes to balance the complexities of the two stages by obtaining an efficient partition of the network into $O(\sqrt{n})$ connected components each of radius $O(\sqrt{n})$. To this end we present efficient deterministic and randomized partitioning algorithms. The deterministic algorithm runs in $O(\sqrt{n}\ \log^* n)$ time and $O(m + n \log n \log^* n) j$ messages, where m and n are the number of links and nodes in the point-to-point part of the network. The randomized algorithm runs in the same time but sends $O(m + n \log^* n)$ messages. The partitioning algorithms are then used to obtain $O(\sqrt{n}\ \log n$ time deterministic and $O(\sqrt{n}\ \log n)$ time randomized algorithms, for computing *global sensitive* functions and a minimum spanning tree.

An $\Omega(n)$ time lower bounds for computing global sensitive functions in both point-to-point and multiaccess networks, are given, thus showing that the multimedia network is more powerful than both its separate components. Furthermore, we prove an $\Omega(\sqrt{n})$ time lower bound for multimedia networks, thus leaving a small gap between our upper and lower bounds.

135. Jan Edler, Jim Lipkis, and Edith Schonberg, ''Memory Management in Symunix II: A Design for Large-Scale Shared Memory Multiprocessors''

While various vendors and independent research groups have adapted UNIX and other operating systems for multiprocessor architectures, relatively little work has been done in anticipation of the software requirements of very large-scale shared memory machines containing thousands of processors. Programming environments for these machines must exploit multi-level memory and cache hierarchies so as to obtain the maximum performance available from the hardware architecture. Several years of experience at New York University with parallel systems and programming environments have led to a memory management design emphasizing flexible control over arbitrarily complex patterns of memory sharing and cacheing. This design integrates user memory management, virtual memory management (swapping/paging), program debugging and monitoring, the file system and buffer cache, and checkpoint/restart in a synergistic manner that extends the existing strengths of UNIX. It is being implemented for two research multiprocessor prototypes, the NYU Ultracomputer and IBM RP3.

136. Jan Edler, Jim Lipkis, and Edith Schonberg, ''Process Management for Highly Parallel UNIX Systems''

Despite early development exclusively on uniprocessors, a growing number of UNIX systems are now available for shared memory (MIMD) multiprocessors. While much of this trend has been driven by the general success of the UNIX interface as an emerging industry standard, experience has shown that the basic UNIX design is amenable to such environments. Relatively simple extensions such as shared memory and synchronization mechanisms suffice for many parallel programs.

While simple needs can be satisfied in a simple fashion, the desire to support more sophisticated applications has created pressure for ever more complex extensions. Is there a better way to meet such needs? Although some argue that it is time to abandon the UNIX model completely, we believe that viable alternatives exist within the traditional framework.

In this paper we propose several modifications to the process management facilities of the UNIX kernel. Some of them are primarily of interest for parallel processing, such as a generalized *fork* system call that can efficiently create many processes at once, while others are equally attractive in other contexts, such as mechanisms for improved I/O and IPC performance. While the primary goals are improved performance and reliability, a strong aesthetic judgement is applied to create a total design that is cohesively integrated.

While the concepts presented here are applicable to any UNIX environment, they have been conceived in the context of very large scale parallel computing, with hundreds or thousands of processors. An initial implementation of these extensions is currently underway for the NYU Ultracomputer prototype and the IBM RP3.

137. Wayne Berke, ''**ParFOR** – A Structured Environment for Parallel FORTRAN''

We present an extension to the FORTRAN language that allows the user to specify parallelism by means of clearly defined, nestable blocks. The implementation achieves compiler-independence through a portable preprocessor. High performance is obtained by *prespawning* processes and relying on a set of run-time routines to manage a self-scheduling allocation scheme. The resulting system, which we call ParFOR, lends itself to the exploitation of fine-grained parallelism because of its low scheduling overhead. It encourages the elimination of explicit process synchronization, thereby enhancing the readability of the source program. In addition, ParFOR provides a variety of modes and compile-time options that are useful for performance measurement and debugging. Finally, we present an evaluation of system efficiency including timing results for several parallel applications running on the eight-processor Ultracomputer prototype.

138. Ora E. Percus and Jerome K. Percus, ‘‘Elementary Properties of Clock-regulated Queues’’

A single clock-regulated queue is fed by Bernoulli streams of messages. We obtain the distributions of queue length, of waiting time, and of output for such a system. This is generalized as well to the case of a queue of finite capacity.

139. Malcolm C. Harrison, ‘‘Add-and-Lambda II: Eliminating Busy Waits’’

This paper describes a further extension of the combining fetch-and-add used in shared memory multiprocessors such as the NYU Ultracomputer and the IBM RPn. This extension permits the elimination of busy-waiting in a number of important parallel operations. These operations include barrier synchronization, stack and queue operations, full/empty bits, and group lock; most of these operations become one or two single-instruction operations for the PE. The hardware necessary is compatible with the design of the switches which are used in the Ultracomputer, and in fact are implemented as fetch-and-add instructions with an increment of unity.

140. Lori S. Grob, ‘‘Automatic Exploitation of Concurrency in C: Is it Really So Hard?’’

The vast majority of work being done today on the automatic exploitation of concurrency, for both multiprocessor and vector machines, is not being done for C. Yet there are many important applications, written in C, which would benefit immensely from the speedup produced by such techniques. Many more applications would be written in C were there optimizing, vectorizing and parallelizing compilers for the language.

In this paper we consider whether it is possible to have compilers that will automatically exploit concurrency in C. We discuss the relationship between automatic exploitation of concurrency for the purposes of vectorizing and multiprocessing. We review the basic techniques and transformations used and examine the necessary conditions to perform these transformations, with examples in C. Several elements of the C language and programming style, such as pointers and recursion, make it difficult to do the necessary data flow analysis. There are two possible approaches to this problem: to bypass code or blocks of code that contain ‘‘difficult’’ features and be unable to apply optimizations to these fragments, or to suitably restrict the language. We examine the choices made by the few available vectorizing and parallelizing C compilers and consider what the future may hold in the light of current research.

141. Yue-Sheng Liu and Susan Dickey, ‘‘Simulation Analysis of Different Switch Architectures for Interconnection Networks in MIMD Shared Memory Machines’’

Differences in switch architectures can have a significant effect on both latency and throughput in interconnection networks, and thus can affect overall performance in shared memory parallel systems with processor to memory interconnection networks. In this paper, we describe several switch architectures which differ in the presence or absence of buffers, in the location and functionality of the buffers, and in the width of the data path.

Since the addition of an architectural feature may have a significant cost in the complexity or cycle speed of a switching component, it is important to have an accurate assessment of the benefit provided. Therefore, we make quantitative comparisons of switch performance using both simulation and analytical methods.

142. Jeanette P. Schmidt and Alan Siegel, ‘‘The Spatial Complexity of Oblivious k-probe Hash Functions’’

We consider the problem of constructing a dense static hash-based lookup table T for a set S of n elements belonging to a universe $U = \{0, 1, 2, ..., m-1\}$. We provide nearly tight bounds on the spatial complexity of oblivious $O(1)$-probe hash functions, which are defined to depend solely on their search key argument. This establishes a significant gap between oblivious and non-oblivious search. In particular, our results include the following:

- Oblivious k-probe hash functions require $\Omega(\frac{n}{k}\ 2e\ -k\ +\ loglogm)$ bits.
- A probabilistic construction of a family of oblivious k-probe hash functions that can be specified in $O(ne\ -k\ +\ loglogm)$ bits, which nearly matches the above lower bound.
- A variation of an $O(1)$ time 1-probe (perfect) hash function that can be specified in $O(n\ +\ loglogm)$ bits, which is tight to within a constant factor of the lower bound.
- Upper and lower bounds for related families of hash functions.

143. Seymour V. Parter and Thomas A. Manteuffel, ''Preconditioning and Boundary Conditions''

Consider the large systems of linear equations $A_h u_h eq f_h$ which arise from the discretization of a second-order elliptic boundary value problem. Consider also the preconditioned systems (i) $B_h^{-1} A_h u_h eq B_h^{-1} f_h$; and (ii) $A_h B_h^{-1} v_h eq f_h,\ u_h eq B_h^{-1} v_h$ where B_h is itself a matrix which arises from the discretization of another elliptic operator. We discuss the effect of boundary conditions (of A and B) on the $L2$ and H_1 condition of $B_h^{-1} A_h,\ A_h B_h^{-1}$. In particular, in the case of H_2 regularity one finds that $\|B_h^{-1} A_h\|_{L2}$ is uniformly bounded if and only if A^* and B^* have the same boundary conditions while $\|A_h B_h^{-1}\|_{L2}$ is uniformly bounded if and only if A and B have the same boundary conditions. Similarly, $\|B_h^{-1} A_h\|_{H_1}$ is uniformly bounded only if A and B have homogeneous Dirichlet boundary conditions on the same portion of the boundary. This latter result does not depend upon H_2 regularity.

144. Jerome Chiabaut, Vladimir Fleyshgakker, and Anne Greenebaum, ''Porting Scientific Applications to the NYU Ultracomputer''

We report our experience in porting six different scientific codes to the Ultracomputer prototype and present promising results on speedups that can be achieved on shared memory MIMD architectures.

145. Olof Widlund, ''Some Domain Decomposition and Iterative Refinement Algorithms for Elliptic Finite Element Problems''

In this contribution, we report on some results recently obtained in joint work with Maksymilian Dryja. We first study an additive variant of Schwarz' alternating algorithms and establish that a fast method of this kind can be devised which is optimal in the sense that the number of conjugate gradient iterations required, to reach a certain tolerance, is independent of the mesh size as well as the number of subregions. An interesting feature of this algorithm is that all the subproblems can be solved at the same time. The method is therefore quite promising for parallel computers. Using a similar mathematical framework, we then consider the solution of elliptic finite element problems on composite meshes. Such problems can be built up systematically by introducing a basic finite element approximation on the entire region and then repeatedly selecting subregions, annd subregions of subregions, where the finite element model is further refined. We consider conjugate gradient algorithms where, in each iteration, problems on the subregions representing finite element models prior to further refinement are solved. This makes it possible to use solvers for problems with uniform or relatively uniform mesh sizes, while the composite mesh can be strongly graded. We remark that this work is technically quite closely related to our previous work on iterative substructuring methods, which are domain decompositions algorithms using non-overlapping subregions.

146. Olof Widlund, ''Optimal Iterative Refinement Methods''

We consider the solution of the linear systems of algebraic equations which arise from elliptic finite element problems defined on composite meshes. Such problems can systematically be built up by introducing a basic finite element approximation on the entire region and then repeatedly selecting subregions, and subregions of subregions, where the finite element model is further refined in order to gain higher accuracy. We consider conjugate gradient algorithms, and other acceleration procedures, where, in each iteration, problems representing finite element models on the original region and the subregions prior to further refinement are solved. We can therefore use solvers for problems with uniform or relatively uniform mesh sizes, while the composite mesh can be strongly graded.

In this contribution to the theory, we report on new results recently obtained in joint work with Maksymilian Dryja. We use a basic mathematical frame work recently introduced in a study of a variant of Schwarz' alternating algorithm. We establish that several fast methods can be devised which are optimal in the sense that the number of iterations required to reach a certain tolerance is independent of the mesh size as well as the number of refinement levels. This work is also technically quite closely related to previous work on iterative substructuring methods, which are domain decomposition algorithms using non-overlapping subregions.

147. Yosi Ben-Asher, David Egozi and Assaf Schuster, ''SIMD Algorithms For 2-D Arrays In Shuffle Networks''

This paper studies a set of 2-D basic algorithms for SIMD perfect shuffle networks. Those algorithms where studied in [1] and [SN], but for the 1-D case, where the size of the problem N is the same as the number of processors P. For the 2-D case of $N = L*P$, we improve some algorithms to be ''2-D efficient'' , by improving their running time from $O(L*\log P)$ to $O(L + logP)$, as N exceeds P. We give nontrivial algorithms for the following operations: Row-Reduction, Parallel-Prefix, Transpose, Packing, Smoothing and Cartesian-Product.

148. Eric Freudenthal and Olivier Peze, ''Efficient Synchronization Algorithms Using Fetch & Add on Multiple Bitfield Integers''

Efficient interprocess synchronization tools for MIMD shared memory computers are critically important for the success of such systems. As members of the Ultracomputer Project at NYU, the authors have been participating in the continuing development of such algorithms. From their work, partially represented in this paper, they have observed that many of the more efficient algorithms utilize the storage and manipulation of more than one state variable in a single machine integer using fetch-and-add (see [GLR 83]). This paper describes this technique and presents several new implementations of busy-waiting synchronization functions which utilize this facility.

149. Edith Schonberg, ''On-The-Fly Detection Of Access Anomalies''

Access anomalies are a common class of bugs in shared-memory parallel programs. An access anomaly occurs when two concurrent execution threads both write (or one thread reads and the other writes) the same shared memory location. Approaches to the detection of access anomalies include static analysis, post-mortem trace analysis, and on-the-fly monitoring.

A general on-the-fly algorithm for access anomaly detection is presented, which can be applied to programs with both nested fork-join and synchronization operations. The advantage of on-the-fly detection over post-mortem analysis is that the amount of storage used can be greatly reduced by data compression techniques and by discarding information as soon as it becomes obsolete. In the algorithm presented, the amount of storage required at any time depends only on the number V of shared variables being monitored and the number N of threads, not on the number of synchronizations. Data compression is achieved by the use of two techniques called *merging* and *subtraction*. Upper bounds on storage are shown to be $V \times N^2$ for merging and $V \times N$ for subtraction.

150. David Wood, ‘‘Parallel Queues and Pools, An Evaluation’’

In this paper I examine a fairly broad range of parallel queue and pool (unordered queue) algorithms and present data indicating their relative strengths and weaknesses. The algorithms have been implemented on the NYU Ultracomputer prototype, an eight processor MIMD, shared memory single bus machine. The data have been normalized to remove the effects of bus contention. I discuss the implementation of a few of the basic synchronization primitives and the implementation of each algorithm to be tested. Two programs were written, one *cqtst* provided a generalized arena for timing the different algorithms under different circumstances. The second program, *fifo*, was used to test the parallel fifo-ness of the queues, that is, to test whether the given algorithm truly represented a queue.

151. Wayne Berke, ‘‘A Cache Technique for Synchronization Variables in Highly Parallel, Shared Memory Systems’’

Caches have traditionally been used to lower the average latency of memory access. When paired with the individual CPU’s of a multiprocessor, they have the additional benefit of reducing the overall load on the processor-memory interconnection. Since synchronization variables have been identified as centers of memory contention, we have looked at methods of utilizing the cache to minimize this effect. A technique of "polling the cache" is proposed to deal with this problem. Consistency within the caches is maintained with a bottleneck-free update facility that exploits the topology of the multistage network. Since as indiscriminate broadcast-on-write policy can lead to severe network congestion at high levels of parallelism, we selectively invoke these updates from the software. We illustrate our methods with a number of useful synchronization algorithms and present simulation results that support the feasibility of our design. In addition to providing support for basic synchronization operations, our methodology is generally applicable to all parallel algorithms that utilize polling.

152. Ronald Bianchini, ‘‘Ultracomputer Packaging and Prototypes’’

This paper reviews interconnection networks and describes packaging of the NYU Ultracomputer. The packaging is based on the ‘button’ technology developed by TRW and adopted by MCC for its ES-Kit modules. This paper presents the basic ES-Kit modules needed to assemble several sizes of Ultracomputers. It also reviews Ultracomputer prototypes and previews future prototypes.

153. Ora E. Percus and J. K. Percus, ‘‘Models for Queue Length in Clocked Queuing Networks’’

We analyze a discrete-time network of queues. The unit element of the network is a 2×2 buffered switch, which we regard as a system of two queues working in parallel. We show how to transform transition probability information from the output of one switch, or network stage, to the input of the next one. This is used to carry our a Markov time series input model to predict mean queue length at every stage of the system. Another model considered is a renewal process time series model, which we use to find the mean queue length of the second stage of the network. Numerical simulations fall within the narrow band spanned by the two models.

154. Anne Dinning and B. Mishra, ‘‘Fully Parallel Algorithm for Implementing Path Expressions’’

Current research efforts in parallel architecture usually evolve from a design of a processor-memory-connection architecture with rudimentary facilities for synchronization and communications, which are often found to be inadequate to support the more sophisticated usages. In order to remedy this problem, we favor a top-down design, where the primitive operations are selected via a careful and thorough analysis of the high-level general purpose applications to be supported by the architecture.

In this paper, we study how a high-level process-synchronization specification, presented in the path expression language, can be translated into a fully parallel implementation on an MIMD shared memory architecture and what primitive support facilitates are necessary to achieve this. Primary emphasis is obviously on the simplicity of the system, low synchronization overhead of the algorithm, efficiency of the primitives and the relative ease of system implementation.

155. Maksymilian Dryja and Olof B. Widlund, ‘‘Some Domain Decomposition Algorithms for Elliptic Problems’’

We discuss domain decomposition methods with which the often very large linear systems of algebraic equations, rising when elliptic problems are discretized by finite differences or finite elements, can be solved with the aid of exact or approximate solvers for the same equations restricted to subregions. The interaction between the subregions, to enforce appropriate continuity requirements, is handled by an iterative method, often a preconditioned conjugate gradient method. Much of the work is local and can be carried out in parallel. We first explore how ideas from structural engineering computations naturally lead to certain matrix splittings. In preparation for the detailed design and analysis of related domain decomposition methods, we then consider the Schwarz alternating algorithm, discovered already in 1869. That algorithm can conveniently be expressed in term of certain projections. We develop these ideas further and discuss an interesting additive variant of the Schwarz method. This also leads to the development of a general framework, which already has proven quite useful in the study of a variety of domain decomposition methods and certain related algorithms. We demonstrate this by developing several algorithms and by showing how their rates of convergence can be estimated. One of them is a Schwarz-type method, for which the subregions overlap, while the others are so called iterative substructuring methods, where the subregions do not overlap. Compared to previous studies of iterative substructuring methods, our proof is simpler and in one case it can be completed without using a finite element extension theorem. Such a theorem has, to our knowledge, always been used in the previous analysis in all but the very simplest cases.

156. Maksymilian Dryja and Olof B. Widlund, ‘‘On the Optimality of an Additive Iterative Refinement Method’’

We consider the solution of the linear system of algebraic equations which arise from elliptic finite element problems defined on composite meshes. Such problems can systematically be constructed by introducing a basic finite element approximation on the entire region and then selecting subregions, and subregions of subregions etc., where the finite element model is further refined in order to gain higher accuracy. We consider conjugate gradient algorithms where, in each iteration, problems representing finite element models on the original region and the subregions prior to further refinement are solved. We can therefore use solvers for problems with uniform or relatively uniform mesh sizes, while the composite mesh can be strongly graded.

The central theoretical issue concerning these so-called iterative refinement algorithms is the design and analysis of algorithms for which the rate of convergence is independent of the number of subproblems, i.e. the number of refinement levels, as well as the mesh sizes. Algorithms for which such bounds can be obtained are called optimal. In this paper, we extend our previous results and show that an additional important algorithm is optimal.

As in our previous studies, we use a basic mathematical framework introduced in a recent study of a variant of Schwarz’ alternating algorithm. Some tools from older work on iterative substructuring methods are also used.

157. Richard Kenner, Susan Dickey and Patricia Teller, ‘‘The Design of Processing Elements on a Multiprocessor System With A High-Bandwidth, High-Latency Interconnection Network’’

The environment of a highly-parallel, shared-memory multiprocessor system with a high-bandwidth, high latency interconnection network is different in many ways from that encountered in a uniprocessor system. We describe these differences and the impact they have on the design of processing elements. In addition, this paper describes methods that can be used to evaluate the impact of architectural choices on the performance of any system that uses a similar network. Two detailed designs of processing elements, one processor and the other using a RISC, are given as examples.

158. Richard Kenner, Ronald Bianchini, Susan Dickey, and Patricia Teller, ‘‘A Compact Design For A Highly-Parallel Shared-Memory MIMD Computer’’

The ultra computer architecture connects hundreds or possibly thousands of processing elements (PEs), each containing a cache but no local memory, to an equal number of memory modules (MMs) via an interconnection network constructed of custom VLSI components that combine (merge) requests from different PEs destined to the same memory location. The network provides a highbandwidth path from the PEs to MMs but the memory latency is significantly larger than that encountered in either a uniprocessor or small bus-based multiprocessor. The PE must exploit the high available bandwidth and minimize the effect of network latency.

159. Eric Freudenthal and Allan Gottlieb, ‘‘Process Coordination with Fetch & Increment’’

The Fetch-and-add (F&A) operation has been used effectively in a number of process coordination algorithms. In this paper we assess the power of Fetch-and-increment (F&I) and Fetch-and-decrement (F&D), which we view as restricted forms of F&A in which the only addends permitted are ± 1. F&A-based algorithms that use only unit addends are thus trivially expressed with just F&I and F&D. Our primary contribution is a bottleneck-free readers/writers algorithm that uses only F&I and F&D. We prove that this readers/writers algorithm is correct and indicate how existing F&A-based algorithms for queues-with-multiplicity can be simplified to yield algorithms using just F&I and F&D. Finally, we discuss a general technique for replacing F&A by F&I/F&D at a cost logarithmic in the number of processors.

160. Steven Jaffe ‘‘Equilibrium Results for a Pair of Coupled Discrete-time Queues’’

A pair or discrete-time queues is coupled by the fact that the arrivals on the trwo are not independent. We derive a functional equational for the bivariate generating function of the joint distribution of queue lengths at equilibrium. This is solved, and the result is used to find exact expressions for the mean and variance of the sum of the queue lengths and to covariance of the pair of the queue lengths. We also find asymptotic formulae for the equilibrium probabilities p_{ij} as $i, j \rightarrow \infty$, as well as some asymtotic results when the arrivval rate approaches 0 or 1.

161. O. E. Percus and J. K. Percus, ‘‘Queue Length Distributions in a Markov Model of a Multistage Clocked Queueing Network’’

In [4], we treated the problem of passage through a discrete-time clock-regulated multistage queueing network by modeling the input time series a_n to each queue as a Markov chain. We showed how to transform probability transition information from the input of one queue to the input of the next one in order to predict mean queue length at each stage. The Markov approximation is very good for $\rho = E(a_n) \leq \frac{1}{2}$, which is in fact the range of practical utility. Here we carry out a Markov time series input analysis to predict the stage to stage change in the probability distribution of queue length. The main reason for estimating the queue length distribution at each stage is to locate ‘‘hot spots’’, loci where unrestricted queue length would exceed queue capacitym and a quite simple expression is obtained for this purpose.

162. A. Greenbaum, ‘‘Parallelizing the Adaptive Fast Multipole Method on a Shared Memory MIMD Machine’’

The fast multipole method is a recently developed scheme for evaluating the potential and force fields in systems of particles whose interactions are Coulombic or gravitational in nature. When the distribution of particles is highly nonuniform, an adaptive version of the algorithm is needed. While the nonadaptive algorithm can be parallelized on an SIMD machine, it is shown that the adaptive version is equally well parallelized on an MIMD machine. Shared memory and an atomic fetch-and-add instruction make implementation especially easy. Results are presented on the 8-processor NYU Ultracomputer Prototype.

163. Anne Dinning and Edith Schonberg ‘‘An Evaluation of Monitoring Algorithms for Access Anomaly Detection’’

One of the major disadvantages of parallel programming with shared memory is the nondeterministic behavior caused by uncoordinated access to shared variables, known as *access*anomalies. Monitoring program execution to detect access anomalies is a promising and relatively unexplored approach to this problem. We present a new algorithm, referred to as *talk*recycling, for detecting anomalies, and compare it with two other monitoring algorithms. Both analytic and empirical results indicate that talk recycling is more efficient in terms of space requirements and performance. Significant conclusions are: (i) Space requirements are in the worst case bounded by O(TxV), where T is the maximum number of threads that may potentially execute in parallel and V is the number of variable monitored. For typical programs, however, space requirements are on average O(V). (ii) Most of the actual performance costs for monitoring is at synchronization operations - the cost per variable access is a small constant. (iii) The general approach of monitoring to detect access anomalies is practical. Various implementation optimizations, as well as the use of complementary tools, can enhance monitoring to effectively solve the access anomaly problem.

164. Allan Gottlieb, ‘‘An Outsider’s View of Dataflow’’

An outsider’s view of the current state of dataflow research is presented. Two related observations are made. First, the field has matured from emphasizing proof of concept to pragmatic concerns needed to obtain high performance. Second, dataflow research is moving closer to mainstream parallel processing. Several examples of dataflow research supporting these observations are given. They include two-stage enabling rules and other eclectic scheduling disciplines, hardware and software throttling of parallelism, non-functional and (possibly) non-deterministic execution semantics, traditional memory management, and von Neumann compatibility.

165. Barry F. Smith and Olof B. Widlund, ‘‘A Domain Decomposition Algorithm Based on a Change to a’’ Hierarchical Basis"

Over the last five years, several new fast algorithms have been developed for the solution of elliptic finite element problems with many degrees of freedom. Among them are Yserentant’s hierarchical basis multigrid method and a number of domain decomposition algorithms for problems divided into many subproblems. The condition number of the relevant operators, for the best of these preconditioned conjugate gradient methods, grows only slowly with the size of the problems; the growth typically is quadratic in the logarithm of the number of degrees of freedom associated with a subregion or an element of the coarsest mesh. Important components of many domain decomposition preconditioners are subproblems related to the sets of variables on the interfaces between neighboring subregions. In this paper, we demonstrate that, for problems in two dimensions, a very simple change of basis for these one-dimensional problems leads to a preconditioner which is as effective as those previously considered. In our proof, we use only tools of linear algebra and a result of Yserentant’s. Our numerical experiments confirm that the new algorithm is better conditioned than the original hierarchical basis multigrid method.

166. Anne Dinning and Edith Schonberg, ‘‘An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection’’

One of the major disadvantages of parallel programming with shared memory is the nondeterministic behavior caused by uncoordinated access to shared variables, known as *access anomalies*. Monitoring program execution to detect access anomalies is a promising and relatively unexplored approach to this problem. We present a new algorithm, referred to as *task recycling*, detecting anomalies, and compare it with an existing algorithm. Empirical results indicate several significant conclusions: (i) While space requirements are in the worst case bounded by O(T x V), where T is the maximum number of threads that may potentially execute in parallel and V is the number of variable monitored, for typical programs space requirements are on average O(V). (ii) Task recycling is more efficient in terms of space requirements and often in terms of performance. (iii) The general approach of monitoring to detect access anomalies is practical.

167. Maksymilian Dryja and Olof B. Widlund, ‘‘Towards a Unified Theory of Domain Decomposition Algorithms’’ for Elliptic Problems"

A distinction is often made between domain decomposition algorithms for elliptic partial differential equations which use overlapping subregions and those which do not. Schwarz’s alternating method, the oldest of them all, belongs to the first category. It has recently been discovered that many of the algorithms that belong to the second category can also be regarded as generalizations of the classical Schwarz method or an additive variant thereof. This new approach provides new tools for the analysis and development of domain decomposition algorithms. In this paper, we introduce an abstract additive Schwarz method and develop a simple framework for the analysis of its rate of convergence. We show that it is possible and convenient to specify algorithms in terms of a set of subspaces and related orthogonal projections. This family of algorithms is further expanded by replacing the linear systems of equations, which correspond to the projections, by suitable preconditioners. We illustrate the usefulness of this approach by considering additive Schwarz methods of a relatively conventional type, iterative substructuring methods, domain decomposition methods defined on the curves or surfaces which subdivide the region, and iterative refinement methods. Throughout we work in a framework of conforming finite elements and self-adjoint problems, but we also mention some new results for more general elliptic finite element problems.

168. O. E. Percus and J. K. Percus, ‘‘Time Series Transformations in Clocked Queueing Networks’’

We consider a network in which message packets arriving on a clock cycle are directed along paths which may have segments in common. When they do, the messages involved are queued and transmitted, one per cycle. The statistics of queue length are investigated by an expansion of the time series in a channel about the regime of independent arrivals. Results for the second stage of such a network are in good agreement with computer simulations.

169. Ora E. Percus and Susan R. Dickey ‘‘Performance Analysis of Clock-Regulated Queues with Output Multiplexing’’ In Three Different Types of 2 by 2 Crossbar Switch Architectures"

Switches in interconnection networks for highly parallel shared memory computer systems may be implemented with different internal buffer structures. For a 2×2 synchronous switch, previous studies have often assumed a switch composed of two queues, one at each output, each of which has unbounded size and may accept two inputs in one clock cycle. Hardware implementations may actually use simpler queue designs and will have bounded size.

Two additional models for a 2×2 switch using queues that may accept only one input at a time are analyzed. The first uses four queues, one for each input/output pair. The second uses two queues, one at each input. In each case, a multiplexer blocks one queue if two queues desire the same output, making these models more difficult to analyze than the previous model. Maximum bandwidth, expected queue length, expected waiting time, and queue length distribution are presented for both models, with unbounded queue size and with queue size equal to 1.

170. Ron Cytron, Jim Lipkis and Edith Schonberg ‘‘A Compiler-Assisted Approach to SPMD Execution’’

Today, two styles of scientific parallel programming prevail. In the SPMD style, all processors execute the same program, with sequential code executed redundantly and parallel code executed cooperatively. In the fork-join style, a sequential thread of control spawns multiple threads to execute a portion of the code concurrently. In this paper we show how to obtain automatically execution efficiency approaching SPMD-style execution for programs written in the more structured fork-join style.

171. Yue-Sheng Liu ‘‘Architecture and Performance of Processor-Memory Interconnection’’ Networks for MIMD Shared Memory Parallel Processing Systems"

In this thesis we study interconnection networks and their switch architectures, the performance of different architectures, under the MIMD shared memory environment, using both simulation and analytical methods.

The networks we study are constructed from a single basic building block, a switch element. A number of alternatives are proposed in designing the switch element. We study the performance of different switches with both analytical methods and extensive simulations. We also propose a switch architecture with multiple hand shaking signals (which is also referred as 2-CTS single switch for a 2 by 2 switch) that gives improved performance.

Various interconnection networks and interested issues related to them are reviewed in the thesis. We propose a new class of interconnection networks called F networks. In comparison to traditional multi-stage network, F networks provide faster communications among nodes within a cluster. Also extra routes available in the F network make the network fault-tolerant.

Under the uniform traffic model, Kruskal & Snir discovered that different stages of buffered omega network experience about the same queueing delay for moderate load. Based on simulations and analysis, Kruskal, Snir & Weiss established a formula for calculating network delays under moderate traffic. From our simulations we discovered that the Kruskal-Snir-Weiss formula holds only for the forward path delay of the networks; the return path delay is actually substantially less than the delay of the forward path. We complete the network performance formula by extending the Kruskal-Snir-Weiss formula to include the return path. We also analyze the network performance under "hot spot" traffic and obtain analytic results on the performance attained.

172. Ora E. Percus and J. K. Percus ‘‘Intrinsic Relations in the Structure of Linear Congruential Generators’’ modulo 2^{β}"

We show two different general relationships that exist among terms of the linear congruential generator

$$x_{i+1} = ax_i + b \bmod 2^{\beta}$$

that are separated by powers of two. These two relationships are not equivalent except in one special case which turns out to be Marsaglia’s result.

173. Patricia J. Teller and Allan Gottlieb ‘‘TLB Performance in Multiprocessors’’

This paper compares the performance, in highly-parallel shared-memory multiprocessors, of locating translation-lookaside buffers (TLBs) at processors with that of locating TLBs at memory. Our performance comparison is based on results of trace-driven simulations of multiprocessors with $\log N$-stage networks interconnecting N processors and N memory modules.

For the systems and workloads studied, memory-based TLBs perform better than processor-based TLBs, provided that memory is organized as multiple *paging arenas*, where the mapping of pages to arenas is fixed. The cost of a processor-based TLB reload is at least $\log N$ because of network transit. The cost of a memory-based TLB reload can be made smaller than that of a processor-based TLB reload, since network transits are not required. Furthermore, with multiple paging arenas, the number of reloads is smaller with memory-based TLBs. For memory-based TLBs to continue to outperform processor-based TLBs for large N, it is likely that the number of paging arenas must grow with N.

174. Torbjörn Granlund and Richard Kenner ‘‘Eliminating Branches using a Superoptimizer and the GNU C Compiler’’

This paper presents a Superoptimizer, GSO, which searches for an instruction sequence that implements a specified operation on one of a set of microprocessors. The basic algorithms used by GSO are described along with the techniques used to allow GSO to be easily configurable to support many microprocesors. We show sequences for generating various comparison operations on the IBM RS/6000 and describe how patterns based on these sequences were added to the GNU C compiler to allow it to generate compact, branch-less code for many common conditional expressions.

Published in this year’s ACM Conference on Programming Languages and Implementation.

175. Susan Dickey and Richard Kenner ‘‘Using Qualified Clocks in the NORA Clocking Methodology to Implement a Systolic Queue Design’’

A systolic queue design has been implemented with the the NORA clocking methodology using qualified clocks. NORA allows the use of compact CMOS circuits with high tolerance for clock skew. Qualified clocks provide a natural way to implement local data movement in a systolic design. Their use with NORA, however, involves certain complications.

A circuit to produce a qualified clock for use in the NORA methodology is presented, and its maintenance of NORA assumptions is proved. Charge-sharing and noise problems which can arise are detailed. The performance of chips we have fabricated using this methodology is described.

176. Guoying Chen ‘‘A Wiring Tool for Ultracomputer’’

During the final stage of computer system design and implementation, using some flexible and convenient tools can save a lot of time and effort. For example, when all subcomponents of a design are carefully placed on a board, and the design schematics and some necessary data are given, a wiring tool is strongly needed so that the wiring can be done easily and automatically. However, it is a practical question how to route each signal wire for *many* components on *any* given board under various constraints and complicated situations.

This report describes a general-purpose wiring tool which meets such a kind of demand, and is used particularly for Ultracomputer prototyping and implementations. The most important part of the wiring tool is how to solve the routing problem for a signal net. We reduce it to a Traveling Salesman Problem (TSP) with various constraints. The wiring tool employs a heuristic and fast TSP algorithm that is able to produce optimal results in high probabilities.

The wiring tool program consists of three parts: *Preprocessor, wiring-man,* and *output generator.* The tool package offers a flexible interface to users, and is able to handle any new coming boards, chips, and other technologies semi-automatically with minimal manual intervention.

In addition to describing the features and the method of implementation of the tool, we also provide useful information and discuss several issues for it’s future upgradings.

177. Ron Bianchini

178. Susan R. Dickey and Richard Kenner ‘‘A Combining Switch for the NYU Ultracomputer’’

This report describes the combining switch that we have implemented for use in the 16 x 16 processor/memory interconnection network of the NYU Ultracomputer prototype. Packaging, message types and message formats are described. Details are given about the internal logic of the two component types used in the network, including the systolic combining queue, the wait buffer and the ALU. Systolic combining queue designs with greater combining capability are sketched.

179. Ora Engelberg Percus and J. K. Percus ‘‘An Expanded Set of Correlation Tests’’

An analytic study is made of the correlation structure of Tausworthe and linear congruential random number generators. The former case is analyzed by the bit mask correlations recently introduced by Compagner. The latter is studied both by an extension to word masks which includes spectral test coefficients as special cases, and then by the bit mask procedure. Although low order bit mask coefficients vanish in both cases, the Tausworthe generator appears to produce a substantially smaller non-vanishing correlation set for large masks—but with larger correlation values—than does the linear congruential.

180. Susan R. Dickey and Ora E. Percus ‘‘Performance differences among combining switch architectures’’

Combining switches can be used in multiprocessor interconnection networks to prevent tree saturation due to hot spot access. Small differences in the capabilities of combining switch architectures can make a significant difference in their performance. We have studied several such architectures using both analytical techniques and simulation.

A new, more accurate method for specifying the queue-length dependent probability of combining is described. This method was used to construct recurrence relations for the queue length probability distribution of four different combining switch architectures. These recurrence relations were used to compute average queue length and output rates; results were validated with simulations.

181. Guoying Chen ‘‘Primary-node Scheme for Cache Coherence in Large Scale Shared-Memory ’’ Multiprocessors"

In this paper we propose a new cache coherence scheme called the *primary-node* method, capitalizing on the observation in [17] that there is usually a primary processor accessing a data block with much higher frequency than other sharing processors (secondary nodes). We associate each memory block with only one pointer that points to the ‘‘primary’’ node responsible for the majority of consistency operations for that memory block, while the remainings are performed collaborated by both secondary nodes and the primary node. As expected, an overwhelmingly large percentage of consistency operations for the block can be performed locally at the primary node, reducing both the access latency and network traffic related to the coherence actions. The overall memory consumption for this scheme is $O(Clog_2 N + Mlog_2 N)$, almost the same as for the chained pointer scheme. Unlike the chained pointers, our scheme eliminates the large latency problem caused by sequential invalidations. Performance analysis shows that our scheme is simple, efficient and scalable.

182. Susan R. Dickey and Richard Kenner ‘‘Hardware Combining and Scalability’’

Scalability refers to the extent to which a system approaches perfect (linear) speedup with large numbers of processors. Fundamental physical laws limit scalability, but in practical terms, bottlenecks due to serialization points at critical sections are often the most immediate limitation on the scalability of an architecture. The NYU Ultracomputer project has developed algorithms without software critical sections that depend on an interconnection network with hardware combining. Such a network has the ability to merge concurrent hot-spot requests into a single memory reference without introducing significant additional delay.

Hardware combining is not without cost, but our experience in implementing a combining switch indicates that the cost is much less than is widely believed. We describe the design choices made in implementing the switch to keep network bandwidth high and latency low. We compare the cost of a combining switch to that of a non-combining switch and discuss the scalability of the design we have implemented to large numbers of processors.

183. Ora E. Percus ‘‘A Multistage Clocked Queueing Network’’

The networks of interest are characterized by synchronous pulsed operation, and are substantially more difficult to analyze than traditional continuous time queueing systems. The control parameter in such a network is the probability ρ of a packet entering a channel per clock cycle; in steady state, with nonsaturating queues, ρ is constant throughout the network. The quantity of practical interest is the distribution of queue length throughout the network, since a queue which does saturate must enter another mode to allow continued operation. It is important to have a concise form for the time series transformation engendered by units of queueing networks.

We treat the problem of passage through a discrete time clock regulated multistage queuing network by modeling the input time series to each queue as a Markov chain. The Markov approximation is very good for $\rho = Ea_n \leq \frac{1}{2}$, which is in fact the range of practical utility, and it compares favorably with numerical simulation.

Since these investigations point strongly to the fact that the near independent ‘‘light traffic’’ assumption is a surprisingly decent zeroth approximation up to half of the saturation flow, we continue by investigating the expected queue length by correlation expansion of the time series in a channel about the regime of independent arrivals.

In both the Markov model and the correlation expansion, the Kruskal, Snir, and Weiss (KSW) conjecture that the average delay at successive stages can be bounded, independently of the network size, is verified.

184. Ora E. Percus ‘‘Why Use Chen-Stein for Sequence Comparison?’’

The bounds found on the distribution of the longest match between two sequences, using the Chen-Stein method, are adequate for long matches but poor for short ones. They are very greatly improved if exact estimates due to Uspensky and others are employed.

185. Ronald Bianchini, Susan Dickey, Gabriel Goodman, Allan Gottlieb, Richard Kenner and Jiarui Wang ‘‘The Ultra III Prototype’’

186. Guoying Chen ‘‘SLiD --- A Cost-Effective and Scalable Limited-Directory Scheme for’’ Cache Coherence"

In this paper, we propose a hybrid scheme, dubbed SLiD ({\m S}calable {\m Li}mited {\m D}irectory), combining the good features of the {\m chained directory} scheme and the {\m limited directory} scheme. Instead of putting two schemes together, we attempt, in the SLiD scheme, to attack the weak points of each individual scheme, namely the performance penalty in the limited directory when pointer overflow occurs, and the linear latency in the chained directory scheme when performing invalidations due to a write. Moreover, the SLiD scheme framework provides designers with three structural options of different performance requirements and implementation cost.

187. Susan Dickey ‘‘Interconnection Network Switch Architectures and Combining Strategies’’

We have studied a variety of switch architectures for use in the multistage interconnection network of a shared memory multiprocessor. In this paper, we describe a taxonomy of buffered switch architectures and give performance results for the different types we have defined. We investigate two techniques for improving network performance in the presence of contention: the use of multiple handshaking signals and the use of message combining. Simulations results are provided for implementation alternatives using switches of varying capabilities, in order to compare the effectiveness of different methods as system size grows. We show that a practical combining switch design, the two-and-a-half-way combining switch, provides performance as good as other more expensive designs.

188. Patricia J. Teller, Allan Gottlieb ‘‘Locating Multiprocessor TLBs at Memory’’

This paper compares the performance, in shared memory multiprocessors, of locating translation-lookaside buffers (TLBs) at processors with that of locating TLBs at memory. Our comparison is based on trace-driven simulations of multiprocessors with log N-stage networks interconnecting N processors and N memory modules.

For the systems and workloads studied, memory-based TLBs perform noticeably better than processor-based TLBs, provided that memory is organized as multiple paging arenas, i.e., multiple clusters of memory modules where the mapping of a page to a cluster is fixed. The cost of a processor-based TLB reload is at least log N because of network transit. In contrast, the cost of a memory-based TLB reload can be smaller, since network transits are not required. Furthermore, with multiple paging arenas, the number of reloads is smaller with memory-based TLBs.

189. Jan Edler ‘‘Molasses: An Ultracomputer Simulator (Preliminary Version)’’

Molasses is a simulator for a complete Ultracomputer system, containing processors, memory modules, and a multi-stage interconnection network supporting combining. The simulated machine is quite close to the Ultra-3 prototype. A large number of parameters may be set by the user to alter the characteristics of the simulated machine and the data to be collected.

Molasses relies on careful implementation, parallelism, checkpoint/restart, and specialized modes of simulation to achieve high performance. Performance may be maximized by automatically building a custom version of Molasses with most parameters fixed to constant values; this allows the compiler to generate superior code, typically doubling performance.

Although the preliminary version of Molasses does not yet simulate the Ultra-3 processors, its good performance and flexibility have already made it useful for network studies.