(1)

ONREUR Report

9-12-C

AD-A218 508

NATO Advanced Workshop on Supercomputing

J.F. Blackburn

27 October 1989

Office of Naval Research European Office

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) 9-12-C	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Office of Naval Research European Office 6b OFFICE SYMBOL (If applicable) (ONREUR)	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Box 39 FPO NY 09510-0700	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
NATO Advanced Workshop on Supercomputing

12 PERSONAL AUTHOR(S)
J.F. Blackburn

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Conference	FROM ____ TO ____	27 October 1989	20

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
25	05	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Communications
Command, Control, and Communications Systems, Norway

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The NATO Advanced Workshop on Supercomputing was held in the Norwegian Institute of Technology, June 19-23, 1989, in Trondheim, Norway. The papers presented concentrated on supercomputer architecture, programming for parallel systems, performance measurements, conversion of sequential programs to parallel programs, and applications programming.

20 DISTRIBUTION AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Ms. Connie R. Orendorf	22b TELEPHONE (Include Area Code) (44-1)409-4340 22c OFFICE SYMBOL 310

☆ U.S. Government Printing Office: 1986—607-044

Contents

NATO Advanced Workshop on Supercomputing

Introduction

The NATO Advanced Workshop on Supercomputing was held in the Norwegian Institute of Technology (Institute), June 19-23, 1989, in Trondheim, Norway; it was attended by 55 participants from Western Europe, the U.S., and Japan. The papers presented concentrated on supercomputer architecture, programming for parallel systems, performance measurements, conversion of sequential programs to parallel programs, and applications programming. Professor J. Moe, of the Institute, gave an opening address in which he spoke of the Institute and its areas of research. This was followed by a brief workshop overview by Dr. Janusz Kowalik, Boeing Computer Services, Seattle, Washington. Summaries of most of the papers presented are given in the following sections of this report.

Supercomputing; Key Issues and Challenges

K. Neves and J. Kowalik, Boeing Computer Services, Seattle, Washington

Traditional Drivers of Computation. Many of the programs that push the limits of computer capacity in scientific computing relate to the solution of partial differential equations. Continuous models are discretized in time and/or space and approximated at grid points. The traditional drivers in this area have been requirements for finer grid structures, higher dimensions, more features, and a higher level of model complexity. However, while these traditional drivers of computation continue, far greater requirements will be generated by solving inverse problems and optimization, cross-discipline analysis, and using new modes of interaction.

Solving Inverse Problems and Optimization. Through the use of large-scale optimization algorithms, scientists can begin to solve the inverse problem of engineering. For example, today we analyze an airfoil by using complex computational fluid dynamics techniques. Quite often the objective is to create a particular pressure gradient over the surface of the wing. Aerodynamicists have learned several heuristics about the desired pressure gradient profiles over the wing, and iterate with the wing design to produce the desired result. With today's algorithm technology, it is possible to suggest the desired pressure gradient and ask the design program to derive the analysis program as an inner loop and solve the problem. However, it is probable that the optimization technique will need to be "steered" by heuristics. The enormity of evaluation of several configuration design problems suggests several orders of magnitude more computer power may be required. As more sophisticated problems can be addressed, there likely will be a role for expert systems technology in the solution process.

Cross-Discipline Analysis. The creation of an advanced aircraft requires structural analysis, computational fluid dynamics, acoustics, control, thermal analysis, and propulsion. These types of analyses must be integrated in the final design. However, science computational programs in these areas are so computationally intense, they are often run separately and the integration is done at the human or department level. With more computing power, the integration could be done in the computing system.

New Modes of Interaction. The advent of advanced graphics enables human capabilities in pattern recognition and visual inference to be given a new window on physics, chemistry, and engineering. The animation of computed results, interactive exploration of volumetric data, systematic techniques for display of multi-dimensional data with dimensions greater than three, have opened the whole field of computing to realistic simulation.

Algorithms. Among the three main contributors to performance in scientific processors--machine organization, the algorithm, and the compiler--algorithmic improvements can be the most effective. They offer the potential for speedups nonlinear in terms of the problem size, speed improvements that increase as we solve larger problems. In vectorized and parallel algorithms, the distribution of and the access to data have decisive impact on computational performance. The study of vectorized and parallel algorithms in conjunction with access to data is fundamental and crucial to the future use of supercomputers. It should include the entire life cycle of the parallel algorithms, which typically includes

- Selecting a generic algorithm
- Developing and coding architecture-dependent algorithm
- Executing on one or more parallel computers, or a simulation on sequential computers
- Measuring performance that includes theoretical and specific explanation for the observed performance
- Demonstration and long-term maintenance of the algorithm in software libraries which may be experimental, public, or commercial.

Several elements of a useful theory of parallel algorithm complexity have been proposed, which are extensions of the current theory of algorithm complexity and account for the time penalties involved in communica-

Dr. Blackburn is the London representative of the Commerce Department for Industrial Assessment in Computer Science and Telecommunications.

tion. These bring us closer to a practical theory of parallel computation but until such a theory is established and practically used, the design of algorithms for supercomputers will be guided by the accumulated experience and commonly applied techniques developed in the past. For example, vectorization deals primarily with reorganizing computation within inner loops, and its objective is to obtain the longest possible vectors eligible for pipelining. Parallel computation, on the other hand, introduces entirely new degrees of freedom for algorithm design. In its most general form, it allows parallel execution of any set of independent tasks whose granularity can vary from a single operation to large subproblems requiring many thousands of operations. The two major styles of computational parallelism are multiple instruction stream, multiple data stream (MIMD) or control parallelism, and single instruction stream, multiple data stream (SIMD) or data parallelism.

The MIMD style requires a programmer to use various decomposition methods known in science and engineering for solving large problems. This style of parallelism is coarse-grained oriented to minimize the cost of communication and requires a global, top-down analysis currently performed by humans. Eventually it may be possible to construct programming tools that can automate to a great extent, the process of extracting parallelism from high-level description of algorithms and mapping these algorithms to specific parallel architectures.

The SIMD style of parallelism exploits simultaneous operations across large sets of data and is appropriate for massively parallel machines with thousands or even millions of processors, as in the Connection Machine or the DAP. The Connection Machine has been applied successfully to molecular dynamics, document retrieval, memory-based reasoning, and in handling a parallel preconditioned conjugate gradient method applied to elliptic partial differential equations reduced to sets of sparse linear equations by finite element or finite difference method.

Another domain that seems to be uniquely suitable for data parallel algorithms, is that of simulating cellular automation on large lattices for obtaining solutions for partial differential equations such as incompressible Navier-Stokes equations.

Machine Organization. To examine supercomputer architecture trends, three categories of parallelism will be used--shared memory, pseudo-shared memory, and distributed memory.

In a typical shared memory system, separate processors share a main memory through an interface-cache, bus, or registers. To overcome the bottlenecks caused by memory contention associated with shared memory systems, many computer designs provide memory local to each computer, while still allowing for a large shared global main memory. This is a pseudo-shared memory design.

The third category is that of distributed memory systems, such as the hypercubes. Connectivity topologies include the N-cube hypercube, grids, nets, rings, and various combinations. In distributed memory systems variation in any one of the following factors can seriously impact algorithm performance:

- Computational capacity of each processor
- Amount of memory of each nodal processor
- Number of processors and memories
- Connect topology of the processors/memories
- Communication speed among processors
- Mode of operation
- System overhead to synchronize the processors.

The three main approaches to the mode of operation are: single instruction/single data stream, SIMD, and MIMD.

Parallelism in Supercomputer Design Today. Supercomputer vendors are currently exploiting a modest amount of parallelism in order to continue to capitalize on the enormously powerful vector processors. This has the least impact on the large body of applications software important to users and vendors. The initial foray into multiple processors in mainframes, minicomputers, and supercomputers has been largely motivated by job stream throughput considerations.

To effectively use more than about four processors, the power of multiple processors must be brought to bear on a single job, rather than simply processing separate job streams. This requires redesigning application programs that were written for sequential machines and the operating system must provide the mechanisms for higher level languages; e.g., Fortran, to create and coordinate tasks within a single job.

Commercially available machines have implemented the same degree of parallelism differently.

1. The Cray X-MP/4 has four processors, each independently programmable, that must contend for main memory

2. The ETA-10 has four processors, each independently programmable, endowed with its own local memory, and sharing a secondary memory

3. The NEC SX-2 has a single processor with four pairs of pipelines and no contention for memory among processors. This is entirely a vector-technology with respect to floating point operations.

Parallelism in Supercomputers--The Next Decade. Because the large stock of existing programs must be used, the major supercomputer companies are taking the evolutionary approach to parallelism. It is likely that through to the mid- to late-1990s, supercomputers will use relatively small numbers of very powerful processors, maybe reaching the order of a hundred or so by the mid-1990s. The pseudo-shared memory approach to operation is likely, with clustering of processors around small numbers of distributed memories. One approach to use

is based on asynchronous (MIMD) use of parallel processors. This would enable inherently scalar (sequential) processes to be parallelized. The other approach is to capitalize on highly vectorized code to enable automatic parallelization at the level of the inner DO-loop.

Compilers and Programming Tools. Techniques to improve performance in vector compiler technology include:

1. Removing unallowable vector forms; e.g., on the ETA-10 and the Fujitsu VP-200, a degradation occurs in vector performance when vectors are not stored contiguously in memory. This can sometimes be obviated by altering the sequence of inner-loop nests. From an algorithmic viewpoint, this can occur when the row of a column stored matrix is required. This can often be avoided.

2. Judicious use of registers, caches, or local memories; e.g., one can alter the sequence of nested loops to reduce the need to store and fetch vectors from memory. This is particularly important when memory access is restricted by memory paths causing a bottleneck. A similar approach can be used when excessive memory fetching and storing causes potential cache misses.

Many applications dealing with physical models have obvious parallel decompositions related to the geometry of the model itself, which cannot generally be exploited at the inner DO-loop level. In moving to a vector or parallel computer, even if the physical model isn't changed, the algorithms used, the mathematical models, and the range of the physical model parameters are often impacted. In an eight-processor vector computer, the combined performance improvement from exploiting both parallelization and vectorization will have a potential of improving performance two orders of magnitude.

With every process that is being carried out in a parallel environment, there is an associated overhead for synchronization. There are two approaches today to parallelizing compiler technology. The first is to exploit the advanced compiler technology to achieve parallel vector optimization. This technique has the advantage of being amenable to automate use of processors through preprocessors, the compiler, and/or other software tools. The second approach to parallelization is to expand the Fortran language by added task creation and synchronization commands that allow the user to structure the computation to take advantage of parallel execution. This approach requires human intervention.

Current Compiler Technology: A Brief Summary. Today's compiler technology, on most, if not all, advanced scientific computers begins with a vector or array dependency analysis. The vectors or arrays are the results of loops common in scientific processing. The objective is to identify operations on arrays such that the input arrays are independent of the result arrays. This will insure that there are no backward array references that would invalidate vectorization or parallelization. Often, loop sequencing or nesting is examined and altered with an eye to creating parallel vector processes. The process is one of mapping Fortran source code to an intermediate set of known constructs such as popular vector instructions. The known vector operations include strided-vector-dyad and triad operations, dot products, and sparse vector operations. More recently, the direct recognition of matrix operations is being employed as well. The dependence analysis techniques employed have become more sophisticated over the years, but often the dependencies cannot be determined by a compiler. At run time, this often requires user intervention.

The automated approach gives easy performance improvement. A user knowledgeable in vector computing can employ the same modeling techniques creating code ideal for the compiler approach.

Compiler technology is seriously lagging behind changes in architecture. Current efforts in compiler technology seem to be concentrated on parallelism based on inner loop techniques. This approach is important but suffers from not exploiting multiple processors at the highest levels of granularity, important in computer designs of the near future; being dependent on data not always available at compile time; and being difficult to manage on a distributed memory system.

Supercomputing at KFA Juelich; Experiences and Applications on Cray X-MP

F. Hossfeld, Kernforschungsanlage Julich GmbH, Federal Republic of Germany

The KFA Juelich is one of the largest big-science research centers in Germany and in Europe. Work ranges from fundamental research to applied science in information technology, material research and solid state physics, and environmental research. Other important fields of research include biotechnology, plasma and fusion physics, nuclear medicine, energy technology, computer science, mathematics and electronics.

The KFA's Central Institute for Applied Mathematics is responsible for the development and operation of KFA's computer networks. Two mainframe computers--IBM 3081/K-64 and IBM 3090/E-128--are hosts to two supercomputers--Cray X-MP/416 and Cray Y-MP/812.

The Cray X-MP Multiprocessor System. Cray X-MP machines are vector computers with specialized pipelined functional units, which can be used in parallel to perform high speed (floating point) computations; and the X-MP series offers the option of multiple processors with shared main memory. The 416 model has four central processor units (8.5 ns cycle time) and a main memory of 16 million words. The special gather scatter and compress index hardware supports vector access to memory using lists of indices.

The four central processors are tightly coupled via the shared main memory and five identical groups of registers, each containing eight shared address registers with

a length of 24 bits, eight shared scalar registers with a length of 64 bits, and thirty-two binary semaphore registers.

The attachment of groups of registers, called clusters, to different processors is done by the Cray operating system COS or UNICOS. One or more clusters can be attached. At present, most Cray X-MP multiprocessor systems are used within a multiprogramming environment, where the individual processors execute different jobs which are totally independent of each other. With multitasking, they may also execute different parts of one program in parallel.

Macrotasking. Macrotasking is multitasking in which parallelism is realized at subprogram level. In order to use macrotasking, the application must be partitioned into a fixed number of tasks, explicitly created by the user through particular subprogram calls. Task management is done by the library scheduler, which obtains information about synchronization and communication requirements through subroutine calls. The library scheduler controls special queues and initiates the necessary activities. The job scheduler assigns the tasks to physically available processors. These tasks can be executed in parallel if there are no prohibitive mutual dependencies concerning correct synchronization and sequencing.

For some kinds of programs, the macrotasking concept leads to an efficient use of the multiprocessing machines. But the user may have to deal with three types of problems that may reduce the speedup achievable by the macrotasking strategy:

(1) The system overhead is too large for small granularity because of the high synchronization frequency

(2) Several tasks cannot be executed in a balanced way on the multiple processors

(3) The number of processors does not match the fixed number of tasks specified within the macrotasking program.

Microtasking. Microtasking is an approach that allows the efficient use of multiple processors even with small granularity. Microtasking also provides parallelization on the statement level, thus providing a different interface to multitasking. Statements, sequences of statements, and complete subprograms can be executed in parallel on multiple processors of a Cray X-MP machine when microtasking is used.

Within microtasking, tasks communicate by means of shared registers which are organized in a cluster. The overhead of the communications via registers is much smaller than the overhead to access corresponding variables in the main memory as realized in the macrotasking library routines.

The microtasking features are provided by preprocessor directives, coded into the user program. This leads to multitasking programs, which are easier to understand than in the macrotasking method. The use of directives does not affect the portability of programs.

Use of Microtasking Control Structures. A microtasking control structure declares a part of a program that must be finished before the execution of the program may be continued. A microtasking process is a sequence of instructions within a microtasking control structure which is always executed by a single task. Microtasking distributes such processes within one microtasking control structure dynamically to the actually available processors. Because of the simple structure of the matrix multiplication algorithm, it is often used to present the different multitasking strategies and to document the potential benefit for such easy to use algorithms.

With the delivery of the new supercomputer Cray Y-MP/832, running the UNIX based operating system UNICOS (in addition to macrotasking and microtasking) autotasking is becoming available that is basically providing efficient means to exploit fine grain parallelism in the multitasking sense. Therefore, one can expect the parallel processing on Cray multi-processor systems (as well as other manufacturers multiprocessors like IBM, Fujitsu, and NEC) will penetrate the supercomputing field. On the Cray Y-MP, new insight will be gained with respect to more massive parallelism by multitasking eight rather than four central processing units.

Supercomputing, the View from NCAR

B. Buzbee, Director, National Center for Atmospheric Research (NCAR), Boulder, Colorado

In two to five years, we can expect an order of 10 increase in computing power and capacity in computer systems. In visualization, we can expect interaction, full color, high resolution, and animation. We can also expect common and rich software and national networks.

The challenge is how to maintain state-of-the-art capability in the face of rapid change in computer technology. We must be able to offer parallel processing, adequate data storage, visualization, distributed computer networks, model development, and technology transfer. The kinds of systems available are coarse grained shared memory systems with powerful vector processors and fine-grained distributed memory microprocessing based systems.

At NCAR, we have access to a Cray X-MP/48 in which a single program can use the entire system for one 2-hour period per day at a favorable cost. We have a model of global ocean simulation with a realistic coastline and it includes a simulation of the arctic current. We have also coupled atmosphere and ocean models by combining existing simulations of each.

By 1990, we expect to have available a 16-processor system with 26 billion words of memory using the UNIX operating system. This system will have five to ten times the power of the X-MP/48 if parallel processing is used.

We have used a Connection Machine that gives the solution of elliptic partial differential equations at a processing speed of 3.8 Gflops on a 4,000*4,000 grid. It was

used to solve explicit shallow water equations. By 1990, we expect to see mainly parallel systems operating at 10-50 Gflops at a cost of around $10 million.

The mass storage system at NCAR contains 53,000 tape cartridges containing 9.1 terabytes. We expect the next mass storage system to have an order of magnitude increase in capacity.

In summary, the next generation of super computer system will have performance measured in gigaflops, storage in giga words, mass storage an order of magnitude greater than today, high volume output, and high quality visualization. The network of the same period will have very high bandwidth and device input/output rates, storage density, and high level of security.

Research and Development in the Numerical Aerodynamic Systems Program

R. Bailey, NASA Ames Research Center, Moffett Field, California

The National Supercomputing Center at NASA Ames Research Center is equipped with a Cray Y-MP, two Cray 2s, eight support systems, and 35 work stations.

The Numerical Aerodynamics Systems (NAS) began in 1984 to provide computing capability for fluid dynamics and to serve as an advanced large scale computing system. The NAS is part of NSFnet and Internet, and provides computing capability for researchers to access from their offices. The NAS, centered in California, has a dedicated line to Langley Field, Virginia, and has satellite connections. The system uses the UNIX operating system and thus has a common user interface.

The NAS has produced research results in fluid dynamics; chemistry; atmospheric science, including climate simulation; and material science. The simulation is mostly steady-state such as the simulation of the space shuttle, which requires up to 30 hours on a Cray 2. Simulations have progressed from two-dimensional (2-D) in 1973 to three-dimensional (3-D) in 1985 and very complex 3-D color in 1989. By 2000, we shall need a computer system able to simulate an entire aircraft or spacecraft system in a few hours.

Some of our current work includes prototyping parallelism in the Navier-Stokes equations, and SIMD on a Connection Machine. We are about to acquire an INTEL system. As data size goes up, there is movement in favor of a Connection Machine versus the Cray 2.

Supercomputing Experience with Operational Parallel Processing

G.R. Hoffmann, European Centre for Medium-Range Weather Forecasting (Center), Reading, U.K.

The Center needs a global model of the atmosphere and sufficient observational data over the entire globe to achieve the best in forecasts. The Center is 60 percent a research organization and 40 percent operational; it has 18 European member states from Norway to Spain.

The Center obtained a Cray 1 in 1980, a Cray X-MP/22 in 1984, and now has a Cray X-MP/48. The geographical spacing for the grid used in the atmospheric model is about 120 kilometers.

The Center is now deciding on a new machine with nine times the power of the present Cray X-MP/48. Included in the evaluation are massively parallel computing systems.

Among the problems are: How best to divide a job between central processing units; verification of forecasts; and portability of Fortran programs between the Cray machine and the IBM3090.

Supercomputing Environment at San Diego Supercomputer Center

S. Karin, San Diego Supercomputer Center, San Diego, California

The Supercomputing Environment at the San Diego Center was established in 1984, and a Cray X-MP/48 was delivered in November, 1985. As of 1988, the facility was used by 250 institutions with usage ranging up to 1,000 hours per user. Users are scattered all over the U.S., and communication is via satellite. The range of disciplines among the users includes physics, biochemistry, atmospheric science, astronomy, oceanography, mechanics, earth science, material science, and industrial applications. Because of interest in superconductivity, there was a big increase in materials science use in 1987-88. Ten percent of the resource is allocated to industry and the rest is for academic research.

Workstations in use by users include Ardent, Sun, and others and network interconnection includes CSnet, NSFnet, BITnet, and others. User services include consulting, documentation, training, applications programming, and data bases.

The essential ingredients for success for such a center are: proven systems, appropriate mission orientation, resource allocation, high-performance communication, an interactive operating system, adequate consulting and training, and appropriate peripheral equipment.

Consulting is handled through a hotline 9 hours per day 5 days per week, electronic mail, and on-site consulting. For training, a 2-day workshop is given several times per month; a more advanced workshop is given in one day several times per month; and a summer institute is run for 2 weeks each summer.

Documentation consists of 130 on-line documents totaling over 16,000 pages, a gather/scatter monthly newsletter, an on-line help package, and 37 bulletin boards. The major current applications work is in biochemistry, chemistry, fluid dynamics, electrical engineering, mechanical engineering, and nuclear engineering.

In the future, we expect to have highly parallel systems with new architecture and large-scale integration. We

also expect to have available new techniques such as optical devices and application-specific graphics. We also need a larger mass storage system and more long trunk communication bandwidth.

Architecture of Fujitsu Supercomputers

K. Uchida and K. Miura, Fujitsu America, Inc., San Jose, California

The FACOM VP2000 series was announced in December 1988, with first delivery expected in March 1990. The largest model--the VP2600/20--consists of a dual scalar processor (clock period 5 ns) and one vector processor with a maximum performance of 4 Gflops. The VP2600/20 has 128 Kbytes of reconfigurable vector registers. Associated with these vector registers are 2 Kbytes of mask registers, with maximum main storage capacity of 2 Gbytes. The VP2000/10 has only one scalar processor.

The VP2400 has half the vector units and performance of the VP2600. The VP2200 has 1 Gflop/s peak performance, 32 kbytes of vector registers, and 512 Kbytes of mask registers. The VP2100 has the same amount of registers but only half the peak performance of the VP2200. The smallest main storage available is 32 Mbytes. All systems can be delivered with or without a second scalar processor, being referred to as model 20 or model 10. Each VP2000 has extended system storage of 1-8 Gbytes and 16-128 channels with a maximum transfer rate of 1 Gbyte/s. The VP2000 system uses chilled water equipment for cooling.

The central processor for any VP system is built from high density (15,000 gates/chip) emitter-coupled logic (ECL), large-scale integration (LSI) chips with 80 picosecond gate delay, and 64-Kbit RAM logic LSI chips with 1.5-nanosecond access time. The main storage consists of 1-Mbit static RAM chips and has an access time of 35 nanoseconds.

Advanced Architecture and Technology of a Supercomputer System

T. Watanabe, NEC Corporation, Tokyo, Japan

The NEC SX-3 Series was announced in April 1989, as the first Japanese parallel supercomputer. The clock cycle time is 2.9 nanoseconds.

The SX-3 will be available in seven models: four multiprocessors and three single processors. The peak performance for the single processors ranges from 1.37 to 5.5 Gflops. The multiprocessor peak performance range is 5.5 to 22 Gflops. The price range for the SX-3 is from $7 million to $24 million.

The 22-Gflop multiprocessor--the Model 44--has four arithmetic processors. The Model 14, the largest single processor has a peak performance of 5.5 Gflops.

In all models, the arithmetic processors use a scalar unit with reduced instruction set computer (RISC) architecture and a scalar pipeline and registers of 128 words of 64 bits. In the vector units there are up to four sets of pipelines: two for addition/shift and two for multiplication/logical operation. There can be two addition and two multiplication pipelines in one set that can be chained for peak performance. The vector registers range from 144 to 36 Kbyte; the mask registers from 256x8 to 64x8.

The system has one to two control processors. One to four input/output (I/O) processors control a maximum of 256 I/O channels giving a maximum transfer rate of 1 Gbyte/s. The main memory size ranges from 64-512 Mbytes in the Model 11 to 1-2 Gbytes in the Model 44. An extended memory unit with a maximum capacity of 16 Gbytes has a maximum transfer rate of 2.75 Gbytes/s. The extended memory unit is also used as an ultra high speed I/O device.

The SX-3 is the first Japanese supercomputer to combine a maximum of four arithmetic processors with shared main memory. The SX operating system has been enhanced for multiprocessor applications by offering an inter-processor communications feature; Fortran 77/SX V2 has been developed as a programming language to provide parallel and vector control statements. The SX-2 had the tools ANALYZER/SX and VECTORIZER/SX and the SX-3 has added the parallelizing tools P-ANALYZER/SX and PARALLELIZER/SX. Up to two control processors can be attached to execute control and general purpose programs.

The SX-3 technology includes ECL chips with 20,000 gates and 70 picosecond switching speed. The high-speed, random-access memory includes both a 40-Kbit memory circuit with access time of 1.6 ns, and a 7,000-gate logic circuit. A new high density multi-chip package allows a maximum of 100 high-speed LSI units to be mounted on a multi-layer, 22.5x22.5-cm ceramic substrate. Each package has a maximum of 2 million gates. The system uses an enhanced, direct water-cooled system for cooling.

Suprenum

K. Solchenbach, Suprenum GmbH, Bonn, Federal Republic of Germany

A Suprenum supercomputer consists of up to 256 processing nodes. Each processing node is a single board vector machine running its own operating system PEACE and communicating with other processing nodes. A processing node consists of

- Node processor (Motorola MC68020, 20 MHz) with paged memory management unit (Motorola NC68851) and scalar arithmetic coprocessor (MC68882)
- 8 Mbyte of node memory (DRAM, 35-nsec static column access time)
- Pipeline vector processor (IEEE double precision) with 2x64 Kbyte of vector memory (SRAM, 20-nanosecond access time)

- Direct memory access (DMA)/Address generation for block transfer of data structure objects
- Communication coprocessor for internode communication.

The node central processor performs the operating system tasks and interprets the instructions of a program. Access to the code and the data objects of the operating system and user tasks residing in the node memory is protected by the node paged memory management unit.

Paging is a means of protection and of providing fast block transfer DMA to the data elements in a page in a static column mode of operation. Because the paged memory management unit adds 45 nsec to a memory access, it is used only on the first entry onto a new page. The pipeline vector processor uses the Weitek WTI2264/2265 chip set-in connection with a microcoded controller accommodated in one of the application specific integrated circuits (ASICs). At 20-MHz clock frequency, the pipeline vector processor has a peak performance of 10 Mflops/s for the single operations. The scalar arithmetic coprocessor provides additional floating point functionality such as conversion, trigonometric, and transcendental. The DMA/address generator allows for a high-speed block transfer of data structure objects. Its microcoded address generators support all required access functions for the data structure types "vector" and "matrix." The DMA/address generator functions are performed by an ASIC.

All four coprocessors use the same unique coprocessor interface of the MC68020, whose functionality has been expanded by a special coprocessor interface (ASIC). There are also ASICs for other functions such as memory error detection and correction, address decoding, data path multiplexing, and bus protocol handling. The ASICs are realized by complementary (symmetry) metal oxide semiconductor gate arrays from LSI Logic, Inc.

The Suprenum Cluster. A Suprenum cluster consists of 20 nodes in one 19-inch rack, of which 16 are processing nodes. The others are the cluster disk controller node, the inter cluster communication node, and the cluster diagnosis node. The nodes of a cluster communicate via the cluster bus system, which consists of two message-switching parallel buses with 64 data lines each. The doubling of the cluster bus is for fault tolerance as well as doubling the interconnection bandwidth.

In the Suprenum supercomputer, the clusters are interconnected via the torus structure, formed by a matrix of bit serial ring buses that transmit data at a rate of 2x125 Mbits/s on the basis of the token ring protocol. The net data rate that the clusters of the torus must share is about 20 Mbytes/s. Inserting not more than four clusters in a ring leaves enough interconnection bandwidth per cluster. Doubling the torus structure by having row rings and column rings not only doubles the interconnection bandwidth, but also renders the structure fault tolerant.

PEACE, The Node Operating System. Suprenum has a local node operating system in each node, while a global operating system exists only virtually, its functions being performed in reality by the collection of node operating systems. The major tasks of the node operating system are: local resource management, including access right control to memory; local process management; and interprocess communication. In performing its tasks, the node operating system may request services from other node operating systems. The node operating system is designed as a multitasking operating system since a highly modularized design of the operating system enhances its efficiency and security and speeds implementation. At the user level, multitasking is a prerequisite for constructing application software independent of the configuration of the machine. The application is partitioned into a number of cooperating processes that are then distributed over the number of nodes of a particular configuration. Process Execution and Communication Environment (PEACE) meets the requirements outlined above. Specifically, it supports the following features:

- Remote access of resources such as files and devices in other nodes
- Remote monitoring of system components in other nodes
- Dynamic reconfigurability of the system after the detection of faults and dynamic reconfiguration for load balancing and service migration in user programs.

The architecture of PEACE is based on the team concept, resulting in a highly modularized, hierarchically structured system. The structures in PEACE are processes and teams. Processes are light-weight processes representing system components that render services to other such components. A team is a group of light-weight processes that share common access domains to intrinsic system objects such as files, memory segments, and processes.

Fast Database Systems

K. Bratbergsengen, Norwegian Institute of Technology

The first prototype database computer developed at the Norwegian Institute of Technology was CROSS8 consisting of nine computers, eight slaves (or nodes) and one master (or host) computer. The master is an IBM-compatible PC/XT and is directly connected via dual port RAM to all nodes. Every node is directly connected to all other nodes. Each node in CROSS8 includes a processor (Intel 80186, running at 8 MHz, giving a computing capacity of about 1 MIP and a bus capacity of 4 MB/s. The main memory is 256-KB Dynamic RAM. The attached Winchester disks are Seagate ST225 PC disks with 21-MB storage capacity and 65-ms average access time.

The successor to CROSS8 is HC16-186, on which work began in spring 1987. Each node of HC16-186 is connected to five modules of shared or dual-ported RAM,

four neighbors, and the host. The system is built as a three-layered cylinder, the center of which is a hypercube interconnection of flat cables, 10 cm in diameter. A nine-cm layer holds the cards of shared RAM. The outer layer, consisting of the node cards themselves, is also about nine cm in diameter. The same processor was used on HC16-186 as on CROSS8 for compatibility, price, complexity, and availability. However the RAM was quadrupled that of CROSS8, and the clock frequency was increased to 10 MHz.

Each module of shared RAM is 2 KB and the bandwidth to the RAM modules is 16 bits, giving a maximum transfer rate of 2.5 MB/s between any two neighbors on the network and a combined transfer rate of 20 MB/s for all nodes when they exchange data with neighbor nodes. The total disk storage capacity is 1040MB and the transfer rate is 7.5 Mbit/s.

Relational Algebra. In a relational database system, the user-specified operations are almost universally translated into a sequence of relational algebra operations. The CROSS8 and HC16-186 are relational algebra engines (see Table 1).

Table 1
Fundamental Operations in Relational Algebra

(1) **Selection**--find records with certain values in certain fields, which is not a matchmaking operation, and is not considered further in this paper

(2) **Projection**--delete given fields of all records in the table; duplicate records may result and duplicates must be deleted

(3) **Union**--two tables are put together and duplicates that may result must be deleted

(4) **Difference**--delete records in first operand that also are found in the second operand

(5) **Intersection**--retain one copy of records that are found in both operands

(6) **Join**--construct result records from records in both operands that have the same value in some given field(s)

(7) **Aggregate**--records with the same value in given fields are grouped together

Selection, projection, and aggregation are one-operand operations, the others have two input relations. The basic method for finding equal records is based on hashing. A hash formula is applied to the operation key and if two records have equal keys, they would hash to the same value.

Records in one relation are spread over all nodes. A hash formula on the primary key gives the node number. When one single record is updated and the primary key is known, only the correct node is activated. This is also exploited every time a new record is inserted when checking for duplicates.

When a relation is searched, all nodes are reading their part of the relation. Selection is easily distributed. Finding equal records is based on repeated use of hashing. The first hash application is for relocating records between nodes. Records that potentially have the same key are gathered in the same node. After this redistribution of records, the completion of the operation is done locally. The hashing technique can be used for reducing search length. Records are placed in several linked lists; the list is again determined by the hash value. The algorithm for doing relational algebra operations in parallel is read records from disk, relocate operands according to a hash value computed from the operational key, complete the algebra operation locally on each node, and write result records to disk.

The Second Hypercube Computer HC16-386. The Norwegian Institute is now working on a hypercube computer based on Intel 80386 as node processor, and it will have an Intel 80387 coprocessor. A minimum of 4 MB of memory is planned for each node. An add-on card for each node processor containing a Weitek WTL 3364 vector processor has been designed, and will give up to 32 Mflops performance.

Thus far, a prototype vector unit has been used for coordinate transformation. With a 16-MHz clock, 32 Mflops are obtained. A program has also been written and run using the vector unit for local matrix multiplication.

Process Scheduling

H. Jordan, University of Colorado

The resources that must be managed include physical devices such as I/O channels and buffer space and shared data items such as primary memory, synchronization delay time, and completion time.

The following items are to be considered in the efficiency of process scheduling:

- Central processing unit is used in each grain, while keeping the overhead of swapping as small as possible; processes should be swapped infrequently.
- Working set in memory is required to use the central processing unit; multi-process working set is required for efficient execution.
- Execution resources are needed for one process for one grain; cooperating processes should be coscheduled over many grains.

A process is an object that can be scheduled for execution. In the UNIX-OS-oriented model, the process structure includes: process status, priority and modifiers, received signal, user identification, central processor and memory usage, controlling terminal, process and parent identification, location and size of swappable image, channel event awaited, and location of text structure. The data segment contains per process data area, the user

structure and kernal stack; user program data; program text; and stack. In addition, the UNIX model contains text segment code and constants.

To ensure coscheduling, enough physical processors should be used and the processors should be assigned as a block and not interrupted. The overhead of asynchronous interrupts on global synchronizations is multiplied by the number of processors. Hardware-supported, fine-grained scheduling should be used. The CDC 6600 Peripheral Processors and the HEP-pipelined, multiprocessor-supported single instruction granularity scheduling in hardware are good examples. The HEP also supports low-cost synchronization waits in hardware. Let the operating system (OS) do the coscheduling required by the parallel program. The simplest mechanism is to identify a group of processes such that no process of the group will be allocated execution resources unless all are.

Multi-Process Parallel Program Interaction with Operating System. Synchronization mechanisms supplied by the parallel language and run-time support are designed to be minimally perturbed by the OS scheduling mechanism. Compiler-extracted or user-specified requirements for coscheduling can be supplied at the start of execution and at other points where the degree of cooperation among the parallel processors changes. Processes of the parallel program may explicitly interact with the operating system at synchronization points: call OS at every synchronization; call OS at types of synchronization expected to wait; call OS after tunable busy wait.

Existing Paradigms for Parallel Program Relation to Operating System. An example of fixed assignment of processes to processors is Flex/32 MMOS, which supports shared memory and synchronization. The Flex/32 MMOS is functionally enhanced by an interface to UNIX running stand-alone on a separate processor.

An example of multiprogramming, OS used for multiprocessing is Encore Multimax OS (UNIX): OS supports memory sharing by disjoint processes; OS supports synchronization by channel events and timed delays; other shared memory synchronizations use busy wait; scheduling is on the basis of I/O compute time ratio; and coscheduling is not available.

An example of two process type with threads running on UNIX processes is given in the first extension of Encore Multimax support: schedulable units are threads, each with private state; low overhead scheduler of threads on UNIX processes; synchronizations interact directly with the thread scheduler; processes interact with UNIX as above; and coscheduling is not available.

In order to determine coscheduling requirements, the synchronization structure of the parallel program must be known. Because synchronization structure can change radically, monitoring of past behavior may be a poor predictor. Also, monitoring disrupts coscheduling which is done by central processors.

Data dependence analysis, as done by parallelizing compilers, can generate information about coscheduling needs. Part of explicit parallel programming is a synchronization scheme planned by the user. The user could base explicit coscheduling requests to the operating system on the operating system.

Coscheduling must be carefully coordinated. An independent OS kernel in each processor can do little without global information. However, a centralized OS sequentializes interactions, which significantly degrades performance.

In the parallel language approach to coscheduling, language constructs not requiring coscheduling are supplied. The barrier is an obvious construct with which to start.

- The purpose of a barrier is to ensure that data dependencies between operations preceding and following it are satisfied.
- Arrival of processes at the barrier ensures that previous operations are complete.
- Reporting of work completion can be done in other ways.

Work barriers can serve the same purpose. The term was used by the IBM EPEX Fortran project specifically for self-scheduled DOALLS; it was extended to a more general setting by Muhammad Benten in his Ph.D. thesis. (He supplied work reporting mechanisms for a range of parallel constructs.)

However, there are drawbacks to the language approach to coscheduling. A general mechanism for reporting work completion by arbitrary parallel operations can be very costly in execution time overhead. Also, simple mechanisms for reporting restrict the use of work barriers.

Some positive aspects of involving the operating system in parallel process scheduling are: parallel programs do I/O also; much I/O remains sequential, even in parallel programs; with parallel extensions, such as disk striping, techniques known from multiprogramming OS will be useful; and groups of processors can be managed as a plentiful resource instead of treating one processor as central to all activity. The OS need not be intrusive. With proper hardware support, the OS can run on processors that are different from those used for parallel program execution.

The key issue of a parallel process scheduling is performance. To efficiently use OS in scheduling parallel programs, don't call OS at every potentially blocking synchronization. There is a low probability of blocking at most synchronizations; they are usually insurance against wrong behavior. Exceptions are major program features (barriers).

Most requirements of a parallel program can be expressed as a number of units of a resource needed simultaneously for efficient execution. The number of coscheduled processes is the main example.

A Cooperative Approach to Program Optimization

J. Levesque, Pacific-Sierra Research Corporation, Placerville, California

There are three questions that I shall address:

(1) Will compilers and/or precompilers be able to automatically generate efficient code for large parallel systems?

(2) Is it reasonable to use previously generated run-time statistics and read the input data at compile time in order to aid in optimization of a Fortran program?

(3) Is it reasonable to ask the user about ambiguous features of the program?

My answer to question (1) is no. There is not enough information in Fortran code at compile time for determination of how best to optimize a time-consuming piece of code; conditional compiling is possible but only useful in limited cases; the hardest task for automatic parallelization is locating the best DO-loop to parallelize; and we can only get code that can achieve at best 30 percent of that possible by hand coding.

On question (2) the input data contains 90 percent of the information needed. Run-time statistics are useful.

The answer to question (3) is yes. However, this implies that programmers will have to understand more about these programs than they do today.

Tools for Parallel Programs

D. Sorensen, Argonne National Laboratory, Argonne, Illinois

For the present and immediate future, we shall be dealing with general purpose computers with shared memory, having synchronization primitives, and a limited number of processors. Vendors are supplying loop-based parallelism featuring micro-tasking, Do-across, and Fork-join. Compiler and precompiler directives are available. These facilities are limited and not sufficient. We need more than simple loop parallelism.

We want portability of programs and to achieve this, we must provide a common interface and programming environment. We would also like to use existing software such as math libraries and applications programs without modification. But there are difficulties: treatment of shared and private common, process creation, synchronization, and interfaces for languages like C and Fortran. The real problem is a lack of standards. A potential programming environment could include globally shared memory, fast synchronization primitives, and some way to create processes.

Among the tools that can be used for parallel programs are:

- Dependency graph to control the flow; this would include data dependencies, execution dependencies and other information
- Memory access pattern program to help in restructuring existing algorithms and devising new ones
- Preprocessor implementation to be applied to the program before execution.

Linear Algebra Library for High-Performance Computers

J. Dongarra, Argonne National Laboratory

In the late 1960s, EISPACK--a mathematical software system for solving eigenvalue problems--was written, consisting of around 50 subprograms for each required precision. The system was really a collection of algorithms rather than a package of software.

In the late 1970s, LINPACK was created, which was a package of mathematical software for solving systems of linear equations. The package also had around 50 subprograms for handling various precisions, used current ideas, and was not a translation

About the same time, the first vector supercomputer--Cray 1--arrived and a set of basic linear algebra subprograms (BLAS) was developed to standardize vector operations. The LINPACK embraced BLAS for modularity and efficiency. Algorithms were reworked and data was oriented by column.

Transfer Rate Compared to Computing Speed. Table 2 shows the peak performance and peak transfer rate to and from memory for several high-powered computers.

Table 2
MFLOPS and Memory Bandwidth

Machine	Peak MFLOPS	Transfer (MW/s)	Ratio
Cray 1	160	80	0.5
Cray X-MP/4	940	1411	1.5
Cray Y-MP/8	2667	4000	1.5
Cray 2S	1941	970	0.5
Cyber 205	400	400	1.0
ETA 10G	644	644	1.0
Fujitsu VP-200	533	533	1.0
Fujitsu VP-400	1066	1066	1.0
IBM 3090/600-VF	798	400	0.5
NEC SX-2	1300	2000	1.5
Hitachi 820/80	3000	2000	0.67
Convex C-210	50	25	0.5
Alliant FX/80	188	22	0.12
SCS-40	44	90	2.0
Ardent Titan 4	64	32	0.5

Increasing the computational power without a corresponding increase in the bandwidth of data to memory can create a serious bottleneck. A single processor of a Cray Y-MP can transfer 500 Mwords/sec and a complete system has a total of 4 Gwords/sec as the peak memory

bandwidth. Another important characteristic of the memory system is the length of time required to access an element of memory, the memory latency. Table 3 shows the memory latency for several supercomputers.

Table 3
Memory Latency

Machine	Latency Cycles
Cray-1	15
Cray X-MP	14
Cray Y-MP	17
Cray 2	50
Cray 2S	35
Cyber 205	50
Fujitsu VP	31

Memory Organization. The memory of a computer system may be vertical or hierarchical. And, of course, fast memories are more expensive. The larger the memory, the longer it takes to access items that were stored randomly. In a memory hierarchy, the register is smallest but fastest. The cache is next, followed by the local memory, the central memory, and finally the secondary storage which is largest but slowest. In such a memory hierarchy, it is essential to keep the active data as close to the top as possible. To partially overcome slow memory, it is interleaved into banks to allow a slow memory to be matched to a fast processor.

Basic Linear Algebra Subprograms (BLAS). In the late 1970s, Level 1 BLAS for vector operations became available, which increased the efficiency and clarity in the programming of vector processors. By the early 1980s, Level 2 BLAS for matrix-vector operations were available, which further increased the efficiency of programming and made better use of memory hierarchy.

By the late 1980s, Level 3 BLAS for matrix-matrix operations became available, which made full use of the memory hierarchy. They provided granularity for parallel processing matrix-matrix operations and made possible parallel operations on distinct blocks, vector and scalar operations to be performed in parallel within subblocks, and full reuse of cache and/or local memory.

Using BLAS freed the implementor to work on more interesting parts of applications. They also increased performance portability of programs across different computing environments; and their use allowed all users to achieve higher performance.

Linear Algebra Library for High-Performance Computers (LAPACK). This package, being developed by professionals at Argonne National Laboratory, Courant Institute, and NAG Ltd, provides a linear algebra library in Fortran 77 language that uses algorithms from LINPACK and EISPACK in a single package. The package runs efficiently on a wide range of high-performance computers and consists of single and double precision routines. The user interface is similar to LINPACK in that it uses a systematic naming scheme and consistent argument order.

History of Block Algorithms for Matrix Problems. Early algorithms in this area dealt with using a small main memory, with tape as secondary storage. Recent work centers on using vector registers, cache, local memory, main memory, or solid-state disks.

An effort is now underway to recast the algorithms from linear algebra in terms of the above Level 3 BLAS; the target machines are the powerful vector processors. The strategy may not be optimal on all machines but is designed to give the best average performance over the range of such high-performance vector processors. And it is expected to perform well over an even wider class of computers.

Maximally Parallel Task Systems

S. Kumar, Boeing Computer Services

The main steps included in the parallelization process are program partitioning into a task system, deriving a parallel task system, scheduling and executing this task system on a multiprocessor system. In this paper, a general framework for an automatic maximally parallel task system generator is presented that may be useful as a component of the code parallelization process. The framework is based on the concept of maximally parallel task systems, a concept that has been mainly used for designing operating systems.

Computational Task System. A task T is an indivisible unit of computational activity specified only in terms of its inputs, outputs, and execution time. We assume that the internal operation of the task is not specified.

A task system C is the pair $C = (J, <)$ where $J = (T_1, T_2, ..., T_n)$ is a set of tasks and is a precedence relation on the set J. The precedence relation defines the operational precedence among the tasks of J. $T_i < T_j$ means that the task T_i is to be completed before the task T_j can start. The ordered pair (T_i, T_j) means $T_i < T_j$..

A task system can pictorially be represented by a task graph. A task graph G of a task system $C = (J, <)$ is a directed graph $G = (J, E)$ with the elements of the task set J as its vertices and the edges in E defined by the precedence relation between the vertices. The edge (T_i, T_j) from T_i to T_j is in E, if (and only if) $T_i < T_j$ and there exists no T such that $(T_i < T_j)$.

A sequential program can be viewed as a task system. The program is segmented into a set of computational tasks along with the given precedence of their execution.

Using the concept of a computational task system, the author showed and illustrated how a maximally parallel task system can be obtained automatically at a reasonable computing effort. The system was determinate and had the minimal number of precedence relations (edges) and

can be scheduled and executed on a multiprocessor computer system.

The Role of Superworkstations in Supercomputing

E. Levin, NASA Ames Research Center

Superworkstations have tightly coupled graphics and computation capabilities. The graphics is 3-dimensional (3-D), realistic, and rapid. The computation should be 1/10 to 1/100 times the speed of a super computer. The workstation should have an adequate main memory and the graphics and computational parts should be architecturally integrated.

The principal vendors of true graphics superworkstations are Apollo, Ardent, Silicon Graphics, and Stellar. AT&T and Pixar have powerful attached processors with excellent graphics. Digital, Hewlett-Packard, Sun, and Tektronix have engineering workstations with graphics capability. The processing characteristics of super workstations include

- Vector, floating point processors - 1-4
- Vector processing speed - 5-30 Megaflops (64 Megaflops peak)
- Integer processing - 10-80 MIPS
- Main memory - 16-128 MegaBytes (access at 300 megaBytes/sec)
- Cache memory - 1 MegaByte (access GigaByte/sec)
- Input/Output - bus bandwidth 80-100 Megabits/sec
- Disk storage - 375-2000 MegaBytes

Software, applications software, and graphics capabilities of super workstations are shown in Table 4.

Graphics standards currently in use include: GKS ISO 2D standard: Programmers High-Level Integrated Graphics System (PHIGS) (ANSI Integrated Graphics Standard); PHIGS + which includes extensions for surfaces, lighting and texturing, and language bindings not well-defined; PEX (PHIGS extended to X-windows, 3-D); RENDERMAN (PIXAR), a proposed standard that separates rendering from geometry; and DORE (ARDENT) for Dynamic Object-Rendering Environment was written in C, is user extensible, is easy to use, and is functionally very powerful.

The 1999 Workstation. The work station of 1999 will likely have ray-tracing quality (radiosity, spatial decomposition); animation (real time dynamics); good integration with supercomputers, other workstations, and networks; sophisticated, easy-to-use software; balance with respect to a supercomputer in processing speed and memory capacity; massive parallelism, with more than 1,000 processors.

The role of the workstation in supercomputing is

- To offload the super computer through stand-alone processing, postprocessing of output, program development, graphics processing, and distributed processing
- To provide scientific visualization for better understanding of results and communication of results
- To provide real-time interaction enabling steering the computation and abort a bad run.

Table 4
Software, Applications Software, and Graphics Capabilities

Typical Software	
UNIX operating system	
Language compilers for Fortran 77, C, and graphics languages	
Extensive networking and interfacing capability	
Applications Software	
Computational fluid dynamics	Electrical CAD
Computational chemistry	Image processing
Geophysical/seismic visualization	CAE/CAM
Structural analysis	Mathematics
Animation/visual simulation	Libraries
Graphics Capabilities	
Screen resolution = 1280x1024	
Image bit-planes including 16-32 bit Z-Buffer and 24 bit color planes	
Display speed 500,000 3-D vectors/sec and 150,000 Gouraud-shaded, Z-Buffered Triangles/sec	
Surface geometry approximations to include polygons, triangular strips, and meshes	
Lighting/shading capabilities: flat, phong, Gouraud shading, ray tracing, transparency, specular highlighting, and texturing support for animation, and stereo	

Issues in Scientific Visualization

B. McCormick, Texas A&M University, College Station, Texas

Visualization is the foremost communications medium in the world for analyzing and describing scientific, engineering, and medical phenomena from the atomic to the anatomic, and is now supported by a computer-based technology.

New features and issues in scientific visualization include visually steering computation; geometry and imaging; volume visualization and modeling; assembly and growth of complex structures; personal, peer, or presentation graphics; and televisualization.

During code development, ninety percent of allotted supercomputer time is currently wasted because of poor interactive visual feedback. Critical aspects of volume visualization and modeling are shown in Table 5.

Table 5
Volume Visualization and Modeling Critical Elements

Representation of complex objects
Knowledge-based modeling uses hierarchical, object-oriented description of parts

Deformable model construction
Deformable 3-D models represent object primitives, which can be deformed for dynamic matching

Dynamic matching
Pseudo-fields deform models for matching with volumetric image data

Constraint-based assembly
Creation of complex models by assembly of primitive object models

Presentation graphics are for conference presentations and journal publications, while peer graphics are for sharing on a timely basis scientific discoveries with colleagues. Throwaway graphics are for scientific analysis by one individual or a small group of individuals.

Televisualization is the study of real-time interactive visualization and modeling conducted over networks that link geographically dispersed investigators. Televisualization has evolved through facsimile networking, picture archiving and communications, hypertext conferencing, 2-D geometric model exchange, 3-D geometric model exchange, animation, volume visualization conferencing, and high-definition TV (HDTV) conferencing.

Much of modern science and engineering cannot be expressed in print at all. Examples are simulated flights through terrain, visualization of flow fields, molecular models, and medical imaging scans. In the 1980's, interactive visual communication in the U.S. is hobbled by lack of standards, mired in the intellectual frustration of making interconnections across incompatible media, and held up at the gateways by disparate transmission protocols never designed with visualization in mind.

Networks that handle screens full of textual information exist and work well. However, the sheer scale of graphics and image data sets exceeds the current bandwidth capacity and interactivity of today's networks. An image that is 512x512 pixels x 8 bits/pixel contains about 100 times more information than a screen of text with 25 rows x 80 characters/row. Volumetric images, approaching 1024x1024x1024 voxels x 4 bytes/voxel, present about 16,000 times as much information as a single 512x512 pixel image.

Semantic image compression/decompression results in cost-effective telecommunications. The process is used to transmit geometric models over common carriers to locally render, display, and animate the transmitted geometric models, and to transmit hybrid geometric models and image representation.

The goal for scientific visualization environment is to provide visualization of 2-D and 3-D imagery, knowledge-based image modeling, image data management, support of multiple image formats, portable user interface, and networking. Object-oriented descriptions provide for the following model classes: command, speech/dialogue, text, window, 2-D geometric model, 3-D geometric model, illustrated document, animation, video, and hyperdocument. Three critical aspects of volume visualization and modeling are (1) navigation in 3-D image data sets, (2) constraint-based assembly/growth of complex 3-D models from a set of basic parameterized objects, and (3) finite element modeling of 3-D imagery.

A scientific visualizing and modeling environment includes topological modeling, geometric modeling, image analysis, computational mathematics, visualization, and user interface management. An interdisciplinary research program in scientific visualization would

- Drive world-class research by scientists and engineers who need both leading edge graphics, image processing, and supercomputing
- Introduce visualization across multiple disciplines
- Build a three-tiered graphical support environment for presentation, peer, and personal graphics
- Develop advanced capabilities for image and graphics support
- Build upon standards to ensure compatibility
- Use televisualization to support collaborators distributed across networks
- Train users and staff in visualization techniques.

Visualization, Engineering, Computers, and CAD

H. Santo, Technical University of Lisbon, Portugal

(Author's Abstract)

With a little exaggeration, there seems to be two main concepts of CAD--from a computer scientist's perspective and from the engineer's point of view. This paper addresses this apparent dichotomy, discusses the inter-relationships between both approaches, and proposes solutions. This paper is also intended to be an assessment of the state-of-the-art and selected source of information of the fields of engineering and the finite element method. The computer science field cannot continue to ignore the outstanding advances achieved by engineering scientists with respect to computational methods and algorithms, with the risk of impoverishing the theoretical foundations of CAD itself.

SARA--A Cray Assembler Speed-Up Tool

R. Babb II, Oregon Graduate Center, Beaverton, Oregon

The SARA (Single Assignment Register Assembler) is an extended form of the CAL (Cray Assembly Language) meant for obtaining near-optimum performance from relatively short Cray X-MP basic block code sequences. The tool, written in Fortran, currently supports

both the X-MP and the Scientific Computer Systems SCS-40. A Cray 2 version is under development.

A SARA programmer has access to all of the facilities of the Cray architecture, and can also treat S, V, and A register as an unlimited supply of "single-assignment" program variables. Single assignment means that while a variable can be assigned a value at most once, the value once assigned can be used many times. In addition, the effect of update-in-place can be specified via the "unite" pseudo-operator.

The SARA optimizing preprocessor (also referred to as SARA) optimizes SARA source files (or pseudo-assembler output from compilers) into a form that is acceptable as input to standard CAL assemblers. Appropriate means are provided to switch between ordinary CAL and SARA in a single program.

The SARA can greatly assist the job of CAL coding from compute-intensive kernels. The preprocessor can automate the complex, tedious, and error-prone tasks of assigning registers and ordering instruction sequences to squeeze the maximum performance out of a particular version of the Cray architecture. The tool can therefore assist in the task of maintaining highly optimized libraries across different models of Crays.

Test results to date compare favorably with expert hand coding, but are much faster to program, and require minimum knowledge of machine- and model-specific timing characteristics, reservations, and conflicts.

Measurements of Problem-Related Performance Parameters

R. Hockney, Reading University, U.K.

Vector (SIMD) Processing. Although the industry has long used the floating point operations per second (MFLOPS) as a single performance parameter, it does not measure performance on a vector or parallel computer architecture because it ignores vector start-up overhead. A more accurate model for characterizing performance uses two parameters to measure asymptotic performance and vector start-up costs. The model uses two more parameters to describe multiple processor synchronization costs and memory bandwidth efficiency. The terms used are: roo is the asymptotic performance in MFLOPS as the vector length goes to infinity; $n_{1/2}$ is the vector length for half maximum performance. The vector length is measured in units of floating point operations lost during the start-up time.

$$p_o = r_{oo}/n_{1/2}, \text{ inverse of start up time}$$

Average performance r in megaflops is given by

$$r = n/t = r_{oo}/(1+n_{1/2}/n)$$

Where n is the actual vector length and

$$t_n = r_{oo}^{-1}(n+n_{1/2}) = po\text{-}1\ (1+n/n_{1/2}).$$

For very long vectors, n is much greater than $n_{1/2}$ and t can be taken as $t = r_{oo}^{-1}n$ and average performance $r = n/t = r_{oo}$. For short vectors where n is much less than $n_{1/2}$,

$$t = p_o^{-1}$$

For average performance $r = n/t = pon = r_{oo}(n/n_{1/2})$. For serial processing $n_{1/2}=0$, for pipeline processing $n_{1/2} = s + L\text{-}1$ where s is the set-up time and L = number of segments; for array processing $n_{1/2} = N/2$ for $n > N$, where N is the number of processors.

Multiple Instruction Multiple Data Stream (MIMD) Computing. Computing with multiple instruction streams introduces additional problems that are not present in traditional computing with a single stream (or processor). The problems are

(1) Scheduling or load balancing E_p (assignment of work or stream (processors) to equalize load and avoid idle processors)

(2) Communication of data routing $f_{1/2}$ (overhead associated with the transfer of data between processors, so that it is in the right place for subsequent processing)

(3) Synchronization $s_{1/2}$ (overhead of ensuring that calculations in different streams (or processors) occur in the correct order and on the correct data).

If a processor is faster than the main data memory, there will be a memory (or communication) bottleneck. If there are f floating point operations per memory reference, then the average time per vector operation is:

$$T = r_{oo}^{-1}(n + n_{1/2})$$

where $r_{oo} = r_{oo}^{a}/(1+x^{-1})$; $f_{1/2} = r_{oo}^{a}/r_{oo}^{m}$ performance ratio

$$n_{1/2} = (n_{1/2}^{m} + n_{1/2}^{a}x)/(1+x);\ x = f/f_{1/2}$$

$f_{1/2}$ is the floating operations per memory reference to achieve half maximum performance, r_{oo}^{a}; $n_{1/2}$ is the problem size required to achieve half maximum performance.

A MIMD program is a sequence of work statements separated by synchronization points. $s_{1/2}$ is the synchronization overhead in units of the asymptotic time per result. r_{oo}^{-1} is the amount of arithmetic between synchronizations to achieve half the maximum performance. The algorithm is:

$$T = r_{oo}^{-1}(s/E_p + s_{1/2}q)$$

where s = total arithmetic in the program, q = number of sequential segments in the program, and E_p = the efficiency of the process use scheduling.

Hockney then illustrated the principles and characteristics he has developed for Cray 2, ETA10, IBM 3090 with VP, and INMOS T800 transputer systems.

Supercomputer Performance Evaluation

J. Martin, T. Watson Research Center, IBM, Yorktown Heights, New York

System architects, software developers, customers, and vendors are all concerned with benchmarking. Their goals and interests intersect, overlap, and sometimes con-

flict. The PERFormance Evaluation for Cost-effective Transformations (PERFECT) Club was organized to develop benchmarks for effective performance evaluation of supercomputers. The senior members of the PERFECT Club are California Institute of Technology, CSRD, Cray Research Inc., Houston Area Research Center, IBM, Institute for Computer Research (Tokyo), and Princeton University. The methodology consists of five stages:

(1) Determine major applications areas and predominant solution techniques

(2) Select a collection of representative programs and a set of target architectures

(3) Define appropriate parameters of the applications and architectures to allow models to be developed

(4) Define metrics and measure performance in controlled environment

(5) Assess relationship between application and architecture models.

Presently, the PERFECT benchmarks represent a collection of scientific applications that have been run on a small number of diverse systems and have provided the basis for developing a methodology for improved benchmarking techniques. The collection of applications is portable public domain codes and is indicative of what a particular set of scientists believe is being done in some of today's high-speed computing environments. The applications are in air pollution, computational fluid dynamics, nucleic acid simulation, structural dynamics, water simulation, seismic migration, quantum chromodynamics, weather simulation, circuit simulation, signal processing, and quantum mechanics. The baseline measurements are wall clock and CPU times, megaflops, best compiler optimization, and profile of dominant algorithms.

Future work will concentrate on

(1) Architectural studies; e.g., why do particular algorithms map to individual architectures, and what opportunities are being missed

(2) Algorithm studies including what are the ingredients of successful mappings and are we ready to generalize

(3) System software to answer--could any of the optimizations performed be automated, and are there general lessons to be learned and applied?

Information Applications on the Connection Machine

D. Waltz, Thinking Machines Corp., Cambridge, Massachusetts

The Connection Machine product family is an associative memory system with a performance of 2,500 Mflops/2,500 MIPS with an array of 65,536 processors, each with its own memory, coupled by a high-speed communications network. In large database applications, for which the Connection Machine is particularly appropriate, individual data elements are stored in separate processors and are operated on simultaneously. The system has 2,048 Mbytes of main memory and 10-640 Gbytes of mass storage and input/output speed is 320 Mbyte/sec.

Three types of applications will be discussed in the use of the Connection Machine: document retrieval, formatting a database, and memory-based reasoning.

Document Retrieval. The document retrieval system mixes artificial intelligence (AI) ideas with methods from information science and is based on a weighted associative memory algorithm. A single Connection Machine system allows a 6-Gbyte free text database to be simultaneously searched and browsed rapidly and conveniently by more than 2,000 users.

Based on a few user-generated keywords, the system returns a list of documents, ordered according to number of keywords, and how important each keyword is (based on how rare it is). The user then browses to find one or more relevant documents. The user may then search for related documents by performing a full text-to-full text comparison between the documents he has found and every document in the database. Specially derived hash functions are used in the search. The method is termed relevance feedback.

Formatting a Database. Formatting a database proceeds in several phases, gradually grouping the individual characters of the raw file into words, phrases, paragraphs, and documents. First an expression-based lexical analysis system is used to break the input stream into tokens; e.g., words and punctuation marks. Then, several dictionaries are used to differentiate important from unimportant words and to group words into known phrases. Finally, groups of words are put into surrogate tables and results are written to disk. A generalized parsing algorithm is under investigation, which may be applied to identifying specialized forms of noun phrases, as well as to full syntactic parsing.

The first step in processing raw text is parsing it into meaningful units. The words in the text must be found and distinguished from other information and the incoming stream of raw data must be split into separate documents with paragraph delimiters, identified headlines, and dates. All of this is done by a regular expression-based lexing phase, which runs the regular expressions by precompiled finite state automata (FSAs).

The lexer is driven off an action list that contains pointers to FSAs, and indications of what to do with the matches. One special directive marks all substrings of the input matching a regular expression as words. Another splits undelimited text into documents while extracting such information as the headline and date.

The text compression algorithm uses word frequency dictionaries, consisting of words compared with how many times they occurred in a corpus of known size. The text compression algorithm being used in the current document retrieval system is called frequency-based index-

ing. The frequency-based indexing extracts content-bearing terms based on the number of times they occur throughout some large database. Words with high frequency such as "the", "of", and "to" are dropped and words with low frequency are retained. Words in between are handled with a word-pairing scheme. An improvement on discriminating among words based on frequency could be parsing based on, for example, multi-word phrases.

Memory-Based Reasoning. Memory-based reasoning is a new paradigm for AI in which an associative memory using a best match algorithm takes the place of rules. In its purest form, memory-based reasoning is reduced to perception. The current situation is observed and it reminds the system of something it has seen before, and an immediate reaction is forthcoming without further analysis.

In heuristic search, solutions are generated rather than sought. In solving a medical reasoning problem, the system finds a patient or patients most similar to the current patient by a global nearest match operation and uses the diagnosis, treatment, and outcome to find a diagnosis and treatment and to predict an outcome. A rule-based forward chaining system takes the patient's symptoms and applies rules one after another until it reaches a diagnosis.

Parallel hardware reverses the relative efficiency of memory-based reasoning and rule-based reasoning. On serial hardware, the best match operation is very expensive because every data item must be considered in turn, while rule triggering is relatively cheap because algorithms exist that allow a database of rules to be efficiently searched. *On parallel hardware, the best-match* algorithm takes constant time, while the rule invocation takes time proportional to the number of rules that must be chained to obtain the answer.

Vectorization and Parallelization of Transport Monte Carlo Methods

K. Miura, Fujitsu America, Inc.

This paper reviewed the computational techniques, the coding methodology, and the performance for the transport Monte Carlo simulation on the vector supercomputers., taking a cascade shower simulation code EGS4 as an example. A reasonable vector performance can be obtained by treating the problem in a different manner from the conventional sequential processing in such a way as to exploit the vector architecture of current supercomputers.

The paper also discussed computational techniques and issues in parallel processing of the transport Monte Carlo codes on the shared memory parallel systems, and compared the vector approach with the parallel approach.

Super Parallel Algorithms

D. Parkinson, Active Memory Technology Ltd, and Queen Mary College, London

The two common routes to parallelism are the vector pipeline and multiple processors. A vector processor has a start-up time (or latency) so the time for multiplying N pairs of numbers is $s + Nt_v$ where s is the start-up time and tv is the beat time for the pipeline. Hence, a first level approximation to the time for performing N repeats of a single operation is $s + Nt_v$. However, in practice, there is a maximum length of vector that any given vector architecture can perform in a single activation. If this maximum is p_v then we must express N as $M_v p_v + q_v$ with $0 < q_v < p_v$. The time to perform the total task is therefore $M_v(s + p_v t_v) + (s + q_v t_v) = (M_v + 1)s + Nt_v$.

If we have a multi-computer configuration with pm processors each with an operation time of t_m then, if $N \simeq M_m p_m + q_m$, the operation time on a multiprocessor is $(M_m + 1)t_m$.

If the speedup caused by parallelism S is the time to compute N results singly divided by the time to compute the results using the parallel facilities, for vector pipeline we have

$$S_v = N(s + t_v)/((M_v + 1)s + Nt_v)$$

which is approximately $1 + s/t_v$ if p_v is $>>1$. Hence, the benefits of the vector pipeline approach are greatest when the start-up time is long compared with the beat time.

For multiple processor systems we have

$$S_m = N_m/(M_m + 1)$$

which is approximately p_m. Because of the qualitative difference in S_v and S_m, vector processing and parallel processing should be considered separately.

In practice, two cases are of interest:

(1) The number of processors is less than the number suggested for the problem at hand. This is the usual case in large matrix manipulations. The increased parallelism will encourage us to attempt to solve larger applications, and matrices of size 200*200 will not be considered especially large. Likely, we shall have access to 200 processors, or at most, several thousand.

(2) The converse of the previous case is also interesting. The size of the matrices that we are dealing with may be only 4*4 and we may have 40,000 processors. The question is now one of embarrassment of riches--how can we use the excess number of processors?

Embarrassingly Parallel. Embarrassingly parallel problems are those where one uses the parallel multiple processors to perform many repeats of a serial algorithm. The problem with such problems is sometimes a need for access to large amounts of data, a requirement which may be unsatisfactory, in real-time signal processing, caused by an effective increase in the latency. The latency time might be unacceptable for the application.

Multiple Parallel Algorithms, Super Parallel Algorithms. Because of the latency problem, we may wish to perform simultaneously several repeats of a given subtask using a given processor set. In practice, the question is "What is the best strategy for solving N repeats of sub-

problem Y using P processors where each of the subproblems has a characteristic size parameter M"? The programmer may wish to write a program that will be portable across a range of processor sizes. For small degrees of parallelism, it may be reasonable to assume that P can change in steps of unity, but for massively parallel systems, P will only change by multiplying factors such as 4.

The problem is characterized by a minimum of three parameters N, M, and P and different algorithms are optimal in different parts of the space:

- If P = 1, we are in the serial regime and the best serial algorithm is optimal
- If P = N, we are in the embarrassingly parallel regime and again the serial algorithm is optimal
- If N = 1, then we are in the classical parallel algorithm design regime
- Special cases exist if P divides N exactly or if P has a 'nice' relationship with M.

The task is that of writing algorithms that solve not just one instance of a problem but solve in parallel many instances of the problem. We call these algorithms super parallel. A typical need for super parallel algorithms arises in solving sets of tri-diagonal equations. This is a common subtask in many problems derived from partial differential equations arising from three-point approximations to second derivatives. Hockney and Jesshope (Parallel Computers 2, Adam Hilger, 1988) is a good source for discussion of the problem of solving m sets of tri-diagonal systems with n equations. The optimal parallel algorithm for solving one set of tri-diagonal linear equations is the cyclic reduction algorithm and uses N processors.

One way of describing the function of a cyclic reduction algorithm is to consider that each step doubles the number of zeroes between the central diagonal and the off diagonal terms. After log2n steps, the off diagonal terms all become zero. If the equations are diagonally dominant, consider replacing the tests versus zero with a test versus some tolerance factor. The code that we have produced contains no explicit reference to the number of equations being solved. This is highly significant.

The code that we have produced will not only solve one tri-diagonal system, it will solve any number of tri-diagonal systems of any combination of lengths, and the number of times that the loop will be traversed is given by the logarithm of the size of the largest system in the set.

Parallel and Vectorized Algorithms for Nonlinear Optimization Problems

D. Conforti, University of Calabria, Cosenza, Italy

This paper deals with the problem of finding an appropriate match between vector parallel supercomputers and nonlinear programming algorithms. The aim of identifying nonlinear programming algorithms, theoretically sound and naturally suitable for vector supercomputing, suggests concentrating on the sequential quadratic programming algorithms.

In the work described in this paper, previous experiments were extended, taking into account other classes of nonlinear programming algorithms and new architectural models of vector-parallel supercomputers. The main purpose was to define and assess a more general framework, on the basis of which it would be possible to study how to exploit and extend the vector parallel supercomputing technique into the field of nonlinear programming algorithms and software.

It is necessary to take into account the following significant points in supercomputing:

(1) A vector-parallel supercomputer is a system characterized by large potentialities in terms of computing speed, memory, and I/O capabilities

(2) Vector-parallel supercomputer use becomes convenient if its capabilities are exploited to take maximum advantage of them

(3) To achieve the above goal, the analysis of the computational characteristics of the algorithms to be implemented is required

(4) On the basis of this analysis, it is possible to define an effective strategy for realizing and/or restructuring algorithms and software suitable for supercomputing.

A typical algorithm for nonlinear programming is structured in two phases: to compute a direction of search and to find a suitable step length for forcing convergence of the iterative process. The first phase is typically implemented as a deterministic subproblem, in the sense that a priori the computational load is known, based on standard algebraic operations. The second phase is strongly affected by the values of the objective function at the current iteration. The second phase is a nondeterministic subproblem, since completely different sequences of calculation might be executed iteration by iteration. The set of computational features of a nonlinear programming algorithm directly depends on the method of implementation of the above phases.

The set of architectural features of vector-parallel supercomputers is easy to define. Number and type of processors, structure of the interconnection network, number and type of vector registers, number and type of functional units, number and type of data paths, and structure and characteristics of the memory hierarchy are typical of the set of features. Each vector-parallel computer has its own set of architectural features. On the basis of these, it is possible to define the set of "ideal" computational features.

Given a nonlinear programming algorithm, the key issue is to try to maximize the intersection between the set of computational features and the set of "ideal" computational features for the specific architecture taken into consideration. This can be done by a suitable modification of the elements of the set of computational features of the algorithm.

In this paper, a minisupercomputer vector-parallel implementation of the sequential quadratic programming algorithm was considered and a computational testing was carried out on large-scale highly nonlinear problems. Experimenting with a new computer architecture with a high level of parallelism had two benefits: (1) evaluation of the various aspects of using a parallel architecture versus a simple vector one, and (2) analysis of the computational features of the sequential quadratic programming algorithm in a parallel computing environment versus the performance. New algorithms have been considered; e.g., a projected Lagrangian method, to assess a systematic view of a broad range of nonlinear programming algorithms and their use and performance in a supercomputing environment.

Large Sparse Linear Systems

I. Duff, Harwell Laboratory, Oxfordshire, U.K.

The Harwell Laboratory is staffed with about 1,400 scientists and engineers; the Computer Science Services Division has a staff of about 120. The laboratory bought a Cray 1 in 1981 and a Cray 2 in 1987.

A typical problem to be solved leads to a system of equations of the form:

$$Ax = b$$

in which A is a very large and sparse matrix. The order of the matrices with which we have dealt has increased rapidly since 1970. Typically, the order n of matrix A was 200 in 1970, 1,000 in 1975, 10,000 in 1980 and 100,000 by 1985.

An important direct method for solving $Ax = b$ for A of high order is the Gaussian elimination method. If we can find matrices P and Q such that:

$$PAQ = LU,$$

where L is a lower triangular matrix and U is an upper triangular matrix, then we may solve:

$$Ax = b,$$

through $Ly = Pb,$

followed by $UQx = y$

The three steps to be followed are: choice of P and Q, factorization forming L and U, and solution using L and U.

Parallelization may be applied at three different levels:: (1) at the system level, use domain decomposition and partitioning; (2) at the matrix level, take account of the effect of sparsity; (3) at the submatrix level, use computational kernels.

Make intelligent and effective use of all the levels of parallelism through parameterizing, exploiting the natural parallelism of the problem, using good algorithms, using kernels, while keeping in mind portability and efficiency. There should be harmony between the algorithm and the system architecture. In a band matrix, using windows can be helpful and for a variable band use changeable, sized windows. An elimination tree can be used effectively for parallel implementation.

Interconnection Networks for High-Speed Parallel Supercomputing

R. Stefanelli, Politecnico di Milano, Italy

This paper treats Multiple Interconnection Networks (MINs) and their adoption in supercomputing architectures. The two main classes of MINs are parallel computer systems implemented by connecting powerful processors and large shared memories, and dedicated supercomputing structures such as high throughput array structures dedicated to signal and image processing that directly implement highly parallel algorithms. For both application classes, adopting fault tolerance methods is important.

General purpose supercomputers have been designed exploiting parallelism in many ways, ranging from SIMD arrays of biserial processors to MIMD structures based on very powerful central processors. Many of these structures are tightly connected, meaning that the data transfer between the connected modules is executed by bus cycles within instruction cycles. Totally different examples of tightly coupled systems can adopt the crossbar switch interconnection. These allow for the highest communication performance, but are characterized by an inherent lack of flexibility whenever the number of processors or memory banks change. The MINs can be defined as interconnection structures obtained by cascading either identical or nonidentical interconnection networks.

Many important algorithms in the fields of signal and image processing show flow graphs that are amenable to the graphs of MINs. This paper analyzes some important approaches to fault tolerance both in MINs and in hardwired, dedicated arrays. The methods considered allow for overcoming production defects.

Interconnection networks can be classified by their communication methods. In circuit switched networks, each node contains at most digital amplifiers that connect input links to output links. These amplifiers can be enabled or disabled to drive their output links. Data moves through a node from an input wire to an output wire through an enabled amplifier, being delayed by gate delays only. Many paths can be contemporarily active between different couples of input and output nodes of the interconnection networks.

In packet switched networks, a node contains a data register and bits of data are grouped into packets as large as a data register. During a communication cycle, a packet is sent from one node, through a link, to another node, where it is stored to wait for the next cycle. In these networks, the links can be either bit-serial; or parallel up to the dimensions of the data registers. In the network, many packets may be routed at the same time. If nodes contain more than one data register, each register can

sustain a communication path, and more paths can be active at the same time and at the same node. A packet train constituted by several packets can be sent in sequence through the network along the same path. A full message-switching communication mode is supported when the packet train can be of arbitrary length.

In both circuit and packet switching, information addressing and controlling the routing of data can be added to the true data by adopting wires dedicated to address bits, or by appa rending a header to the message data, or a control packet to a train. This information can be used by the control algorithms implemented at the nodes, in order to set up a communication path, by selecting what amplifiers must be enabled or what following node the packets of a train must be sent to. These mechanisms allow for the dynamic creation of communication paths through the network.

Well-known examples of interconnection networks are rings, meshes, trees, crossbar switches, and n-cubes. This paper treats the interconnection networks whose structures are represented by directed graphs called banyans that are particular forms of Hasse diagrams. A Hasse diagram ia an intransitive, irreflexive, and asymmetric directed graph. If two nodes are connected by a path passing through some other nodes, they are not directly connected by an arc e.g., intransitive. Irreflexive means that no node has a self loop and asymmetric means that if a path goes from node a to node b, then no path comes back from b to a. A Banyan is a Hasse diagram such that for all s_i source nodes and for all s_j sink nodes there is exactly one single path from s_i to s_j. Various kinds of useful Banyans have been defined. Some of them show regularity and some show rectangularity. The various Banyan definitions are provided in Table 6.

Table 6
Banyan Definitions

A Banyan is regular if all the nodes (except inputs) have the same number of incoming arcs (in-degree), and all the nodes (except outputs) have the same number of outgoing arcs (out-degree).
A Banyan is called l-level if every path from inputs to outputs is of length l. For these Banyans it is possible to group intermediate nodes into intermediate levels; i.e., level i is the set of nodes at the end of all the paths of length i starting from input nodes. A binary tree is an example of an l-level Banyan.
A Banyan is rectangular if there are the same number of nodes at each level.

Fault-Tolerant Interconnection Networks for Parallel Processors. The networks discussed in this section are circuit-switched networks providing a complete communication within a set of N processors constituting the input nodes of the network and a set of N memories constituting its output nodes. The primary goal of such a network is to grant that any input node be reachable from any output node. The secondary goals are to achieve the highest parallelism of communication while keeping the cost and the complexity of the interconnection network within acceptable bounds. For simplicity, any switch need only perform the following actions: recognize its own setting command in the message header; set its input-to-output connections correspondingly; and delete its own setting command from the message header and forward the remaining message along the network.

A variety of fault models have been devised. Stuck-at faults have been assumed for both control and signal lines, so that a switch is either incapable of assuming one of its positions (stuck-at fault on control line), or incapable of transmitting correct data on one of its data lines (stuck-at fault on an input line).

The fault tolerance criterion for multistage interconnection networks can be summarized as providing multiple paths between input and output terminals so that total connectivity can be kept even in the presence of suitable fault distributions. One method of reaching this goal is by introducing additional links, rather than additional stages. An alternative approach is to add a row, rather than a stage, to the basic structure.

The four main philosophies concerning fault tolerance of MINs are graceful degradation, added stages for redundancy, added rows of switches or added links, and added complexity of the switches and of related control.

Fault-Tolerant Interconnection Networks for Dedicated Supercomputing. Particularly significant cases to consider are processing elements are simple arithmetic circuits as in the case of the Fast Fourier Transform; processing elements are central processing units or computers; processing elements are packet-switch processors.

Assumptions that can be made regarding the fault model are multiple faults are allowed for in processors, links, and redundant circuits and links required by the reconfiguration algorithms; all the faults are considered at the functional level; and a faulty element, whether processor or link, is completely unusable.

Two approaches to fault tolerance based on time redundancy were discussed in detail and are briefly described here.

(1) In inter-stage reconfiguration, the redundant links and circuits allowing for time redundancy-based reconfiguration apply strictly to the topology of homomorphic computational graphs. (A computational graph is obtained by interconnecting the basis iterations and exploiting both the temporal and spatial parallelism.) The reconfiguration algorithm does a search based on a highly structured set of rules and carries out the required reconfiguration.

(2) In intra-stage reconfiguration, the geometry of the interconnection networks between each stage, and of the

computational graph, is not related to the design of the redundant links and circuits allowing for reconfiguration. In this approach, the reconfiguration algorithm searches for time spare(s) of a faulty processing element inside the same stage where the fault has been located. Not only does the reconfiguration proceed stage by stage, but the only information necessary for reconfiguration is related to the stage under reconfiguration.

Reference

Hockney, R.W. and C.R. Jesshope, *Parallel Computers 2*, (Adam Hilger, Bristol, 1988).