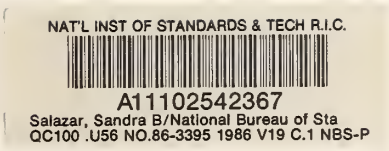


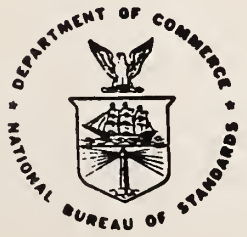
NBSIR 86-3395

National Bureau of Standards Workshop on Performance Evaluation of Parallel Computers

Sandra B. Salazar
Carl H. Smith

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

July 1986



U.S. DEPARTMENT OF COMMERCE
ONAL BUREAU OF STANDARDS

NBSIR 86-3395

National Bureau of Standards Workshop on Performance Evaluation of Parallel Computers

Sandra B. Salazar
Carl H. Smith

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

July 1986

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 86-3395

NATIONAL BUREAU OF STANDARDS WORKSHOP ON PERFORMANCE EVALUATION OF PARALLEL COMPUTERS

Sandra B. Salazar
Carl H. Smith

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

July 1986

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

NATIONAL BUREAU OF STANDARDS WORKSHOP ON PERFORMANCE EVALUATION OF PARALLEL COMPUTERS

Sandra B. Salazar and Carl H. Smith *

U.S. Department of Commerce
National Bureau of Standards
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

The Institute for Computer Sciences and Technology of the National Bureau of Standards held a workshop on June 5 and 6 of 1985 in Gaithersburg, Maryland to discuss techniques for the measurement and evaluation of parallel computers. Experts from industry, government and academia presented position papers and reported on experiences with parallel computers. This document is the report of the workshop.

Key words: parallel processors; parallel architectures for computers; performance evaluation; benchmarking.

Foreword

The Institute for Computer Sciences and Technology at the National Bureau of Standards is actively engaged in the development of techniques to measure and evaluate the performance of parallel computers. As a preliminary step, a workshop on performance evaluation was held in Gaithersburg, Maryland on June 5^{th} and 6^{th}, 1985. The goal of the workshop was to define the issues and problems involved in the development of benchmarks for large parallel computers. Thirty-six talks were given by representatives of government, industries, universities and research laboratories. The topics presented ranged from specific measurements of large parallel machines to the philosophical issues concerned with the development of universally applicable benchmarks. In addition to the formal talks, there were several lively discussions.

This document is a report on the workshop. A one-page synopsis of each presentation is included. The synopses were compiled from notes taken and copies of the speakers' transparencies. Each contributor had the opportunity to review the synopsis of his or her presentation. Pointers to particular synopses are bracketed in slanted type face. Many of the synopses mention commercial products. These references in no way constitute an endorsement by the National Bureau of Standards or any of its employees.

---

* Also affiliated with the Department of Computer Science at the University of Maryland and the University of Maryland Institute for Advanced Computer Studies

Introduction

In the past decade, the cost of conventional computers has dropped dramatically while the processors themselves have become significantly faster. Although most of the advances are due to enhanced fabrication technology rather than innovative architectures, it is widely believed that further fabrication improvements are unlikely to yield major performance gains. Thus the unending search for faster computers has led scientists to emphasize new computer architectures. In particular, parallel architectures have received considerable attention [*Connoly*, *Schneck*].

With these new parallel architectures comes the problem of evaluating them. Standard benchmarking techniques display several shortcomings when applied to conventional sequential machines. All the same criticisms apply to the benchmarking of parallel computers. In addition, new problems arise when trying to benchmark parallel machines. Some of the difficulty lies in the diversity of the architectures under consideration. Language constructs used to exploit multiple instruction stream, multiple data stream (MIMD) architectures are radically different from constructs designed for use on a single instruction stream, multiple data stream (SIMD) machine. Consequently, a benchmarking program written for one type of machine may not run on any other parallel machine without extensive modifications altering the very nature of the algorithm underlying the benchmark. In any event, the performance of a benchmark may fail to predict how the machine will behave when asked to run application programs written by the intended users of the machine.

The participants of the workshop all agreed that benchmarks for large parallel computers were necessary to influence the design decisions of vendors and for the procurement of machines [*Hayes*]. Academic comparison of machines would be enhanced by a reasonable set of benchmarks and the National Bureau of Standards was deemed an appropriate, unbiased collection point for such a set [*Nelson*]. The participants also agreed that there were several pitfalls to avoid when benchmarking parallel computers. Unfortunately, there was no consensus on how to avoid those pitfalls. In some cases, there weren't even any ideas on how to solve aspects of the benchmarking problem. The remainder of this section is devoted to listing problems that will be encountered when trying to benchmark parallel computers.

The benchmarking of standard sequential computers is complicated by word length differences. If the benchmark uses 40 bit aggregates of data, then a processor will take between 1 and 5 memory accesses to get one logical word of data. A relatively slow processor might run a benchmark faster than a "fast" processor with a narrow memory bandwidth. Consequently, a benchmark designed to assess processor speed might actually (when run on several different machines) measure memory efficiency.

The word size problem is amplified when discussing the benchmarking of parallel computers. By their very nature, parallel computers are designed to process several data items simultaneously. Some of the data must come from external sources (disk files) as input, and some data is correspondingly designated as output. Incoming data must be transferred from an essentially serial device and loaded into an appropriate memory location before processing can begin. The reverse process is necessary for output. Perhaps the processors can process data faster than it can be received as input or fed to the output device. Interconnection network topology and bandwidth influence input/output speed

[*Arnold, Bailey, Faiss*]. The former factor is evident only on parallel architectures. Again, the question as to what is actually being measured arises. Separating processor performance from input/output proficiency may well be impossible, and perhaps undesirable, on some parallel architectures. Similarly, separating processor performance from benchmark overhead is trickier for parallel architectures than for inherently sequential ones.

If a benchmark is to be useful, the same program must run on several different machines. Consequently, assembly languages exploiting unusual characteristics of individual architectures are ruled out. Two fundamental problems stem directly from the use of high level programming languages for benchmarking. Firstly, compilers vary tremendously as to the efficiency of the code they produce. Consequently, a benchmark may actually measure compiler efficiency as opposed to processor performance [*Allen*]. There is no widely used standard language for parallel processors. All parallel processors use an extension of some programming language developed for sequential processors. Each extension is designed to exploit particular features of the host machine. Consequently, a benchmark must be "translated" into the dialect used for each machine. In some cases, the translation is not straightforward, involving modifications to the underlying algorithm. Consequently, the benchmark may measure the quality of the translation and not machine performance.

The problems with benchmarks written in high level programming languages are compounded by the applications bias inherent in today's parallel processors. Each architecture has features designed to facilitate particular parallel operations [*Browne, Jordan, Schwetman*]. The different features are incarnated in the extensions to the standard base language. It would be misleading, and in some cases unfair, to use a benchmark designed to exploit a particular type of parallelism on a machine geared toward a different type of parallelism. The classic example is SIMD vs. MIMD. However, no one type of parallelism seems to be common to all parallel architectures. So, in addition to other problems with benchmarking, each result must be qualified by some notion of the appropriateness of the benchmark algorithm for the the particular computer.

Designs of parallel architectures sometimes call for thousands, even millions of processors [*Uhr*]. Prototypes with far fewer processing elements are constructed for program development and benchmarking. Studies done with 4, 8 and 16 processor versions of the same general architecture indicate that the results for one number of processors do not extrapolate to a larger set of processors [*Clark, Lakshmirvarahan, Thomas*]. Hence, even if the above problems with benchmarks could be solved, the results obtained from prototype machines would be virtually useless in evaluating production models.

Current parallel computers are notoriously difficult to program. Benchmarking should address the issue of productivity throughout the complete process of algorithm design, implementation and use. It may well turn out that a particularly fast processor is so cumbersome to program, that a solution to some problem may be achieved more rapidly on a standard sequential computer that is significantly easier to program.

The result of a benchmark is generally a single numeric value [*Nelson*]. A single value cannot possibly sort out the issues of raw speed, input/output speed, net productivity and ease of use. This is especially true for parallel architectures where overall efficiency must be traded off versus total time to task completion. Single numbers are sought after by administrators, not versed in the technical issues, who seek simple metrics for the purposes

of procurement [*Dembart*].

The above comments do not represent the views of any one individual. Rather they are a compendium of the views expressed by the participants of the National Bureau of Standards Workshop on Performance Evaluation of Parallel Computers. Several of the above issues were hotly debated during the workshop.

Since the workshop, an on line library of benchmarks has been set up at the National Bureau of Standards. The library is accessed by sending electronic mail to:

`NBSLIB@ICST-CMR.ARPA`

Sending the message:

`send doc from info`

will cause a brief description of how to access the library to be returned to the sender. The library contains several well known benchmarks (e.g. whetstone and dhrystone) and some benchmarks developed at the National Bureau of Standards. There are currently 78 benchmarks in the collection. Additions are welcome.

List of Attendees

Walid Abu-Sufah - University of Illinois
George B. Adams III - NASA Ames Research Center
Randy Allen - Rice University
Clifford N. Arnold - ETA Systems, Inc.
David H. Bailey - Informatics General Corporation for NASA/AMES
Robert P. Blanc - National Bureau of Standards
Vito Bongiorno - Cray Research, Inc.
Martha A. Branstad - National Bureau of Standards
Al E. Brenner - Fermilab
Richard Brice - MCC Corporation
Stephen A. Brobst - Massachusetts Institute of Technology
Jim C. Browne - University of Texas
William E. Burr - National Bureau of Standards
Robert J. Carpenter - National Bureau of Standards
Ronald S. Clark - IBM/Kingston
John W. D. Connolly - National Science Foundation
James E. Cottrell III - National Bureau of Standards
Kent K. Curtis - National Science Foundation
Benjamin Dembart - Boeing Computer Services
Jesse M. Draper - National Bureau of Standards
Kenneth M. Dymond - National Bureau of Standards
Rudolf O. Faiss - Goodyear Aerospace Corporation
John B. Freeman - Control Data Corporation
D. Odell Hamilton - National Bureau of Standards
Ann H. Hayes - Los Alamos National Laboratory
Arthur W. Holt - National Bureau of Standards
Harry F. Jordan - University of Colorado
Malvin Kalos - New York University
Robert M. Keller - University of Utah
William A. Kneisly - ETA Systems, Inc.
Alfred L. Koenig - National Bureau of Standards
James T. Kuehn - Purdue University
S. Lakshmivarahan - University of Oklahoma
Olaf Lubeck - Los Alamos National Laboratory
Gordon E. Lyon - National Bureau of Standards
Creve Maples - Vitesse Electronics
Joanne L. Martin - IBM/Yorktown
Ernst Mayr - Stanford University
Kevin P. McAuliffe - IBM/Yorktown
Alan Mink - National Bureau of Standards
Harry Nelson - Lawrence Livermore National Laboratory
Rodney R. Oldehoeft - Colorado State University
John P. Riganati - National Bureau of Standards

Steve Ritzman - National Bureau of Standards
John W. Roberts - National Bureau of Standards
Sandra B. Salazar - National Bureau of Standards
Paul Schneck - National Security Agency
Herb Schwetman - MCC Corporation
Zary Segall - Carnegie-Mellon University
Carl H. Smith - National Bureau of Standards
Robert H. Thomas - BBN Laboratories
Michael Tsao - IBM/Yorktown
Leonard Uhr - University of Wisconsin
Walt Vandevender - Sandia National Laboratory
Robert G. Voigt - NASA/ICASE
Shukri A. Wakid - National Bureau of Standards
John Zelenka - ETA Systems, Inc.

SPEAKERS AT THE NBS WORKSHOP ON PERFORMANCE EVALUATION

Enhancing Performance of Small Grain Multiprocessors
Walid Abu-Sufah - University of Illinois

A Performance Study Involving Designers and Users
George B. Adams III - NASA Ames Research Center

Evaluating Program Transformations for Supercomputers
Randy Allen - Rice University

Performance Implications of a Hierarchical Memory in a Multi-processor System
Clifford N. Arnold - ETA Systems, Inc.

Memory Bank Contention on High-Speed Computers
David H. Bailey - Informatics General Corporation for NASA/AMES

Performance Tools for CRAY Computers
Vito Bongiorno, Jr. - Cray Research, Inc.

Past, Present, and Future Uses of Computers in High-Energy Physics
Al E. Brenner - Fermilab

Performance Evaluation of the Tagged Token Dataflow Architecture
Stephen A. Brobst - Massachusetts Institute of Technology

A Universal Simulator for Parallel Computation
Jim C. Browne - University of Texas

VS FORTRAN Program Multitasking Facility Application Performance Analysis
Ronald S. Clark - IBM/Kingston

The NSF Supercomputer Initiative
John W. D. Connolly - National Science Foundation

Allocating Costs in a Multiprocessor Environment
Benjamin Dembart - Boeing Computer Services

Impact of Operand Types on Machine Performance
Rudolf O. Faiss - Goodyear Aerospace Corporation

Benchmarking Activities and Plans at Los Alamos
Ann H. Hayes - Los Alamos National Laboratory

The HEP Multiprocessor and the Principle of Universal Parallelism
Harry F. Jordan - University of Colorado

The NYU Ultra Computer: Its Architecture and its Applications
Malvin Kalos - New York University

The Rediflow Simulator
Robert M. Keller - University of Utah

Evaluating the Performance of the PASM Reconfigurable SIMD/MIMD Machine
James T. Kuehn, H. J. Siegel, and Leah H. Jamieson - Purdue University

Experience with HEP
S. Lakshmivarahan - University of Oklahoma

A Benchmark Comparison of Three Supercomputers: Fujitsu VP-200, Hitachi S810.20, and CRAY X-MP/2
Olaf Lubeck - Los Alamos National Laboratory

The Importance of Performance Analysis Tools in Highly Parallel Architectures
Creve Maples - Vitesse Electronics

Analysis and Measurement of Scientific Applications Codes
Joanne L. Martin - IBM/Yorktown

Nonparallelizable Algorithms
Ernst Mayr - Stanford University

Performance Monitoring of the Research Parallel Processor Prototype (RP3)
Kevin P. McAuliffe - IBM/Yorktown

Why NBS Should Set the Computer Classification Standards
Harry Nelson - Lawrence Livermore National Laboratory

Performance Considerations in HEP SISAL
Rodney R. Oldehoeft - Colorado State University

Plans for the NSA Supercomputing Research Center
Paul Schneck - National Security Agency

Parallel Workloads
Herb Schwetman - MCC Corporation

Programming and Instrumentation Environment for Parallel Processing
Zary Segall - Carnegie-Mellon University

Performance Measurements for the Butterfly Parallel Processor
Robert H. Thomas - BBN Laboratories

Performance Monitoring of the Research Parallel Processor Prototype (RP3)
Michael Tsao and Kevin McAuliffe - IBM/Yorktown

Toward Architectures that Execute Algorithms Efficiently
Leonard Uhr - University of Wisconsin

Benchmarking at Sandia National Laboratory, Albuquerque
Walt Vandevender - Sandia National Laboratory

Early Experience with the FLEX/32
Robert G. Voigt - NASA/ICASE

Synopses of Talks

Enhancing the Performance of Small Grain Multiprocessors*

Walid Abu-Sufah
University of Illinois

A taxonomy for depicting the evaluation issues for high-speed computers shows the great diversity of data structures, program structures, and machine structures which must be addressed in performance evaluation.

Performance evaluation activities at the Center for Supercomputing Research and Development at the University of Illinois include:

* machine instrumentation for performance measurement, monitoring and analysis
* performance enhancement techniques

An example of performance enhancement techniques can be found in compiler optimizations for reducing data synchronization in multiprocessed loops. A hierarchical approach to multiprocessing involves:

* large grain
  - decompose the problem into large tasks, each to execute on a computational resource (e.g. a multiprocessor with a small number of processors, each with vector instructions, and a fast network)

* small grain
  - multiprocessing within a task (subroutine).
  - data synchronization requirements of loops are due to inter-iteration data dependencies, such as flow dependence and output dependence

By using our data synchronization reduction technique, synchronizing references to some shared data items, might eliminate the need to synchronize references to many other shared variables.

Other issues of concern are forward and backward branching, compile-time questionable data dependencies, and enforcing of output dependencies. Some results of using our techniques on the 61 subroutines in Eispack are:

* only 38 subroutines need inter-iteration synchronization in the multiprocessed loops

* the number of lock variables was reduced by more than 10% in all subroutines

* synchronized memory references were reduced by 50% for 1/2 of the subroutines

*Supported in part by the National Science Foundation under Grant No's. DCR-84-10110, and DCR-84-06916, the U.S. D.O.E. under Contract No. DOE-DE-FG02-85ER25001.

A Performance Study Involving Designers and Users

George B. Adams III
Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, California

Progress in concurrent processing research has been limited by a lack of understanding of interactions between computational models, system architecture, and classes of algorithms. A goal of the Research Institute for Advanced Computer Science project is to develop usable systems.

In September 1984, a two-week study of the Massachusetts Institute of Technology static data flow machine and the language VAL focused on algorithms for six applications of interest to NASA Ames Research Center (computational fluid dynamics, computational chemistry, galactic simulation, linear systems, queueing network models, and natural language processing). Seven one- or two-person teams of scientists studied data flow concepts and mapped their applications onto the machine, coding in VAL. The goal in mapping was to maximize the number of busy processing elements and to minimize the traffic on the routing network.

The scientists utilized terminals which made the RIACS computer system, running the VAL translator/interpreter, appear familiar to them. The problem-solving steps were as follows:

Problem statement (prose)
Mathematical model (mathematics)
Abstract algorithm (discrete mathematics)
High-level program (VAL)
Machine program (machine language)

This experimental study provided a number of valuable findings, including the following:

* VAL is readily learned

* The data flow model of computation is readily learned

* Most applications could use all parallelism available in the data flow machine

* Small data flow machines may be quite cost effective

Evaluating Program Transformations for Supercomputers

Randy Allen
Rice University
Houston, Texas

An effort is underway at Rice University to review existing FORTRAN programs for application to vector or parallel machines. Parallel algorithms have been found to run slower on parallel machines than was anticipated by either the users or the machine designers. The compiler tends to get the blame for this. Compilers are needed which can use the parallel features effectively. There is also the "dusty deck" problem involving old existing programs which need to be transformed for parallel machines. In the case of new programs, a user may choose to program directly in vector code, although this may turn out to be formidable. Imposing a sequential order among a block of vector statements requires the same analysis as vectorizing sequential statements.

Program development and debugging may be done on a different machine than the one that runs the object program. In this case, there are portability problems to be considered. The transformation process needs to optimally adapt to each architecture on which the program is to be run.

There is a problem in finding suitable programs in the public domain. Artificial benchmarks may include code to deliberately trick the compiler.

Two main tools are in use at Rice University for this work. One is the Automatic Vectorizer and the other is PTOOL (Programming Tool). These are useful for both vector and parallel supercomputers. Effectiveness is evaluated by actually running the code on a specific machine. The amount of computer effort to perform transformations is also noted, as well as the situations in which the transformations are useful.

Performance Implications of a Hierarchical Memory in a Multiprocessor System

Clifford N. Arnold
ETA Systems

The characterization of large machines by means of a single number is becoming less realistic. Solutions are difficult to visualize. CPU speed is proportional to memory speed and capacity (Amdahl's Law) and is limited by the slowest component. Performance issues include the ratio of memory speeds, ratio of memory capacities, memory management algorithm, and software overhead.

The Working Set size (WS) in comparison with the memory size is a factor in performance. For example, consider a system with a memory hierarchy -- a small fast memory (M1) and a large slow memory (M2). When WS is greater than M1, performance is proportional to the performance of M2. When WS is similar in size to M1 and migrates around a larger static user space, performance is a compromise of the performance of M1 and M2, and is determined by the hit rate of the application, the memory manager algorithms and overhead, and the performance of M1 and M2. When WS is close to the size of M2, the system is dominated by the application; the application should be optimized so as not to adversely impact other users of the system.

Only when the static user space is less than the size of M1 is the hierarchical memory not an issue. If M1 is shared by multiple CPUs, then an application with WS proportional to M1 dominates these CPUs whether it uses all of them or only one. This encourages multitasking for the sake of system throughput. If M1 is large enough for several users' WS, then the CPU can efficiently support those several users simultaneously. This is important for quick interactive response and for high throughput of many short and intermediate duration jobs. If the memory hierarchy is managed by virtual memory managers in the OS, then the user can enjoy the capacity of M2, with the performance of M1 without writing I/O.

In conclusion, performance assessment is becoming more difficult. CPU based performance kernels are probably meaningless, as memory organization is becoming more important. Some system designs encourage (or require) multiprocessing. The effective working set of a code on a particular design can only be discovered by running the code on that machine, testing both its hardware and software.

Memory Bank Contention on High-Speed Computers

David H. Bailey
Informatics General Corp.
for NASA/AMES

One cause for reductions in the performance of a high speed computer system is memory bank contention -- the delays encountered when one CPU attempts to access a memory bank that is busy processing a previous access by another CPU (or even the same CPU). This problem will become more pronounced in future generations of supercomputers because of the growing imbalance between CPU speed and memory operation speed.

Two models of memory bank contention were presented. The simple model can be solved exactly using techniques from Markov chain theory, yielding the conclusion that the number of memory banks n necessary to preserve a constant level of efficiency is approximately proportional to the number of processors m times the square of the bank reservation time t.

The advanced model has these assumptions:

* At each system clock tick, every free CPU inititiates, with probability r, a vector memory access (fetch or store).

* Starting bank numbers for vector accesses are uniformly distributed on {1,2,...,n}.

* Vector lengths are uniformly distributed on {1,2,...,V}, except that a larger fraction v of the lengths are the maximum value V.

* Vector strides are uniformly distributed on {1,2,...,n}, except that a larger fraction s of the strides are 1. The term stride refers to the memory increment between successive words fetched or stored.

The more advanced model has not been solved by analytic methods; however, it can be analyzed using Monte Carlo simulation methods. Simulations run on a Cray X-MP/48 produced results consistent with the simple model analysis. The number of banks necessary to preserve a constant level of memory efficiency is approximately proportional to the number of processors minus one times the square of the bank reservation time.

These studies suggest that unless much faster memory chips are developed, future generations of computers will need to employ very large numbers of independent memory banks.

Performance Tools for CRAY Computers

Vito Bongiorno, Jr.
Cray Research, Inc.

A variety of software tools exist on CRAY computers to help the user understand and optimize programs. These tools cover a wide spectrum of user needs. For example, there are static analysis tools that provide the user with information on the references to FORTRAN variables and vectorization potential at compilation time. In addition, there are a number of dynamic analysis tools that allow the user to analyze program components such as subroutines and do-loops during execution. Several of these tools use a unique hardware feature of the CRAY X-MP called the hardware performance monitor.

The hardware performance monitor contains a set of eight counters to track hardware related events that can be used to indicate performance. Some examples of events that can be tracked are the number of instructions issued, numbered central processor hold cycles, fetches, scalar or vector memory references, floating point adds, multiplies, and reciprocals. In total there are 32 hardware events that can be tracked. The performance monitor counters operate in parallel with the execution of a user's program and do not interfere or delay the events being monitored. Two of the software interfaces to the monitor, one a job control language statement, and the other a FORTRAN compiler option, allow its use without any program modifications. In the case of the FORTRAN compiler option, statistics are generated on a subroutine basis, and include the central processor time, floating point operations, the average memory reference rate, the ratio of memory references to floating point operations, and the average number of floating point operations per second (MFLOP) rate.

Through the use of the hardware performance monitor and supporting software, the user can identify important subroutines, determine if they are executing at a high MFLOP rate, and concentrate optimization efforts there.

Past, Present, and Future Uses of Computers in High-Energy Physics

Al E. Brenner
Fermilab

In particle physics, a typical experiment may involve 10^5 interactions per second with 10^6 bytes of data per interaction. The nature of the computational problem in particle track analysis is basically simple, in spite of the data volume, but vector machines have not been successfully employed for this purpose so far. Currently under development is a machine with 128 nodes which will be completed in the summer of 1985. The nodes are all supplied via a branch bus which is felt to be a limiting factor. The Hypercube machine appears interesting but has more capability than is required for this application.

Performance Evaluation of the Tagged Token Dataflow Architecture

Stephen A. Brobst
Massachusetts Institute of Technology

The tagged token dataflow architecture (TTDA) is based on a dynamic dataflow model. It is an MIMD machine, with each processing element highly pieplined. The criteria for evaluation include an assessment of the price/performance ratio as compared to conventional supercomputers as well as scalability. Difficulties with performance evaluation of this architecture arise from the complex interactions between the compiler, the machine architecture, and resource management policies.

To attack this problem, both simulation and emulation are being employed. As input, both methods take dataflow graphs generated by the compiler for the ID language. In cooperation with IBM Yorktown, a detailed simulation of the TTDA has been developed. This software implementation of the machine facilitates experimentation with low level details of the machine architecture. It collects complete timing and performance data on the dynamic behavior of the machine during program execution. In stand-alone mode on an IBM 4381, the processing rate is 20 million dataflow instructions per day.

Emulation of the TTDA will permit evaluation of the architecture on the basis of more substantial application codes than simulation can support. Currently, the emulaton comprises 8 LISP machines connected via a 10 megabit ethernet. The final configuration will consist of 64 LISP machines with some packet switched communication network. A circuit switch has been implemented as our first step toward a high-speed interconnection network. The packet switch will follow in approximately one year. The inter- connection network is easily reconfigurable and heavily instrumented to facilitate performance monitoring.

The TTDA emulation is built on top of generic multiprocessor emulation software. This package provides primitives for:

* defining the number and type of processors being emulated
* specifying the interconnection topology
* monitoring the program execution
* handling the control panel interface

The user must define each logical processor in the architecture to be emulated as a LISP program which exhibits the behavior of the physical processor by modifying local data structures (state) and communicating to other processors via message sending primitives.

The LLNL SIMPLE code has been translated to ID and compiled into the base language (a dataflow graph) for the TTDA. This code is being run on the simulator to assess the machine resources required by this application.

A Universal Simulator for Parallel Computation

Jim C. Browne
University of Texas

The problem addressed in this talk is how to design computation structures and predict performance of computation across architectures, given the diversity of models of parallel computation, as well as the diversity of parallel architectures. The basis for the universal simulator is to develop a unified model of parallel computation, a declarative programming language, and a program synthesis process. The declarative specification includes:

* schedulable units of computation
* dependency relations
* relations among dependencies

All computations can be structured as directed graphs in which the schedulable units of computation form the nodes, while the dependency relations constitute the arcs. This type of model features a clean separation of concepts and an abstract machine specification, as well as a declarative specification.

VS FORTRAN Program Multitasking Facility
Application Performance Analysis

Ronald S. Clark
IBM/Kingston

The VS FORTRAN program multitasking facility allows 2-way and 4-way parallelism for single FORTRAN jobs executing on various large multi-processor systems: 3090, 3081/3084, and 4381-3. Experiments have been run to test the improvement in throughput time for various degrees of parallelism. Modification to the FORTRAN code to achieve parallelism consists of utilizing forks in the program which fan out to multiple tasks which can be run concurrently. Upon task completion the separate tasks then rejoin under the main task. One example is the dispersing of an array among the various available processors.

Three sample applications were modified for parallelism:

1) TBLADE - turbine blade analysis, using a finite difference algorithm

2) VA3D - a fluid dynamic application, using Euler approximationn

3) BOAST - an oil reservoir simulation, using iterative line successive over-relaxation techniques

The majority of the parallelizable execution time was spent in a subset of subroutines. The lines of code run in parallel was 19%, 36%, and 24%, respectively, for an effective parallelizable execution time of 83-93%. Note: Since the Workshop, IBM announced the 3090 Vector Facility (10/1/85) which does support parallel execution of vector code on 3090 model 200 and model 400 configurations. However, all results presented at the Conference pertained to scalar code. Comparable measurement results for parallel vector will be published by IBM.

The observed speed-ups over a single processor were a factor of about 1.7 - 1.8 for two processors and about 2.6 - 3.3 for four processors.

The NSF Supercomputer Initiative

John W. D. Connolly
National Science Foundation

Supercomputers and networks of supercomputers form the leading edge of the "megatrend" from an industrial to an information society. The primary purpose of the NSF is to "help meet the long-term needs of the country by ... the support of science and engineering research and the training of future scientists and engineers." The NSF Supercomputer Program will meet these needs by establishing a network of supercomputer centers for academic basic research. The NSF supercomputer initiative will:

* Make a significant impact on research in the U.S., allowing the solution to many unsolved problems.

* Train students and new scientists in the use of advanced computers.

* Stimulate the innovative parts of the computer industry.

The NSF Office of Advanced Scientific Computing (OASC) was established in 1984 to provide immediate access to supercomputers. It now provides time at six separate commercial and university supercomputer centers. Time is being distributed to more than 300 research groups. Four new centers dedicated to the support of NSF basic research will be started within the next year.

Four sites for the NSF Supercomputer Centers have been chosen as a result of a nationwide competition. These are: San Diego (USCD campus), Illinois (U. of Illinois, Urbana), John Von Neumann Center (near Princeton), and Cornell. The NSF budget is 10 million dollars per year per center, plus cost-sharing with states and industry.

Future plans include:

* Networking
  - Expand the existing community networks, such as the ARPAnet, BITnet.
  - Pilot projects to test technology for a nationwide network, involving satellites, earth stations, and fiber optics.

* Software and Computational Mathematics
  - Development of algorithms, languages, compilers, operating systems, etc.

* Experimental Machines
  - Access to prototypes of future supercomputers.

Allocating Costs in a Multiprocessor Environment

Benjamin Dembart
Boeing Computer Services

The goal of this project was to improve utilization of a multiprocessor Cray system used for commercial timesharing. A billing algorithm was developed, based on the idea that reducing job costs would improve system throughput and resource utilization. Charges were based only on resources used, not on the environment. Multitasking was encouraged since inactive CPU's are inefficient.

The model developed has these characteristics:

* Billed Time = Job CPU Time/Efficiency

* Efficiency =
  System CPU Time/(Wall Clock Time * No.Processors)

* Efficiency depends on the job mix

* CPU Fraction =
  SUM(CPU Time)/(SUM(CPU Time) + SUM(Channel Time))

* Resource Utilization Efficiency (RUE) is the efficiency measured with the system loaded with identical jobs. It is dependent on the CPU fraction and the number of tasks.

Another approach considers the RUE as a function of memory:

* Effective Task Memory =
  Job Memory * Task CPU/Job CPU

* RUE depends on CPU Fraction and Effective Task Memory

* Task Billed Time = Task CPU/Task RUE

* Job Billed Time = SUM(Task Billed Time)

Testing was based on a random job mix, with 11 cases run on a one processor system and 13 cases run on a two processor system. In all cases the billed time covered the wall clock time and in some cases exceeded it by 30%. It was concluded that the resulting billing algorithm would provide an incentive to users to increase their use of multitasking.

Impact of Operand Types on Machine Performance

Rudolf O. Faiss
Goodyear Aerospace Corporation

Problems posed by humans exhibit much diversity. It is no wonder, then, that in responding to problems, man encounters and treats a rich variety of information types. During digital problem solving sessions, the different information types are treated by using different kinds of information elements. Each element is defined by a "clump of bits" of varying bit count and form, i.e., by an information entity invented by man.

Typical entities encountered include:

a status bit,
a one bit logical variable,
a 16 bit 2's complement number,
a 64 bit CRAY floating point,
an 8 bit character,
a list of any of the entities above,
a set of lists,
etc.

A "higher order" entity is one that is likely to be described in terms of "lower order" entities. Thus, a binary number is a "higher order" entity than the individual "place" bits of the number, a list of numbers (vector) entity is of higher order than the components (numbers) of the list, a set of lists (e.g., matrix) entity is of higher order than the components (vectors) of the list, etc.

While an "ideal" 1 bit processor (a processor that executes instructions in a vanishingly small time, has memory with near zero access time, and has near infinite bandwidth channels) provides the needed versatility to handle the myriad entities of any arbitrary kind of problem solution algorithm, it doesn't exist.

When early users of 1 bit processors were confronted by the fact that such processors were painfully slow when executing arithmetic on N-bit scalar entities, they rejected the call for a multiprocessor comprising a set of N one bit processors. Instead, they designed a processor biased toward treating N-bit "scalar" entities. In the processor, all N bits of a scalar are treated in a unified manner by a common controller; memory accesses and channels are laid out in parallel to support such processing. The processor provides the original example of Single Instruction, Multiple Data (SIMD) stream architecture! The new architecture outperformed the original architecture by N:1 when treating the N-bit scalar entity, but performance ability degraded substantially when entities treated were the more primitive 1 bit entities.

Generally, although more than one kind of entity is treated and/or developed while a processor executes a problem solution algorithm, a real digital processor makes more efficient use of its hardware resources (memory, channels, processing unit(s), control, etc.) when treating specific high order entities. A VAX 11/780 does not like to treat CDC 7600 floating numbers. CRAY machines are architecturally biased to treat 64 bit floating point number entities. Other machines (e.g., the Goodyear MPP) are biased toward treating higher order entities such as lists.

In general, processors designed for treating low order entities can treat high order entities with equal efficiency (but slowly); processors designed for treating high order entities treat low order entities with substantially reduced efficiency. When a high speed processor is used, it should be used within its sphere of applicability.

The NBS ought to identify the performance of a machine in terms of its ability to treat a large number of low to very high order information entities; a vector of speed ratings (corresponding to an input vector of entity types) is required. The potential user would be obliged to establish the frequency of use of each entity in his applications environment. In conjunction with the NBS vector, the user would then have sufficient information to establish a "best" speed rating for a mono, multi, or mixed processor computing system. How close the user could get to the best rating is out of the hands of the NBS; the NBS could not guess at what processor lashups users might employ. (At best, the NBS could caution users about lashup pitfalls.)

The development of the appropriate entity set would be a major NBS task. As an example, it would be shortsighted to limit the entity set to only one vector entity comprising 64 bit floating point components. To solve surveillance and AI problems, vectors comprising 1 bit components are common (and, perhaps, most common). The NBS ought to recognize that its close association with the scientific community tends to sanctify the 64 bit floating point number; the NBS will need to exert special care so that the association does not distort its rating efforts.

At best, a single speed rating number for a processor is dangerous to use as the basis for selecting a processor. The NBS should resist efforts to describe processor performance with a single number.

Benchmarking Activities and Plans at Los Alamos

Ann H. Hayes
Los Alamos National Laboratory

The computer benchmarking effort at LANL is being conducted for measurement and comparison purposes, for procurement purposes, and as a means of influencing vendors. There is a concentration of interest at the intersection which occurs among mathematical models, programming languages, and realizable architectures. The machines of interest are vector processors, parallel architectures, and innovative architectures. Benchmark programs are typically subsets of production codes; these production codes may involve 150 thousand or more lines of code. LANL adheres to the rule that productivity of people is first, and productivity of machines is second. The benchmark tests are designed to measure raw machine speed, hardware-specific features, and compiler influence. There is a further need to examine graphics and I/O.

The Los Alamos benchmark set is representative of the laboratory's workload, including Monte Carlo, particle-in-cell, hydrodynamics, and reactor safety problems. The Livermore loops are also used. Recent machines tested include the Cray XMP/24 and XMP/48, Fujitsu VP200, Hitachi S810, and the ELXSI. Experimental computers studied were the Denelcor HEP and the Intel iPSC (Hypercube). Pitfalls to be aware of are:

* benchmarks are valid only for the tested workload (with typical vector length of 50)

* results are time-dependent

* results are influenced by the machine configuration.

Of concern in the testing of multiple processors is the throughput of a single processor versus that of the aggregate, as well as the choice of the best algorithm for a particular configuration versus portability.

The HEP Multiprocessor and the Principle of Universal Parallelism

Harry F. Jordan
University of Colorado

The focus of this paper was on multi-processors of the MIMD type, on large-scale parallelism in the range of hundreds to thousands (but not infinitely many) processors, and on single problem solution speed rather than multiprogramming. Six "Articles of Faith" were proposed:

* Multiprocessing is the way.
* Shared memory should be supported by hardware.
* Multiprocessor algorithms should be independent of the number of processes.
* Process management should be done at the very top of the program hierarchy.
* Performance can be predicted very accurately on the basis of simple models.
* Performance measurements show fine structure which is qualitatively obvious.

User mapping of problems to memory structures is difficult. Consider programming the Illiac IV or the Hypercube. Architectural techniques are available to support shared memory in the near term. These include: instruction stream pipelining that removes dependence on round trip time; message switched processor-memory interconnection with a high step rate; Ultracomputer style combining switch that reduces congestion to a manageable level; and local cache memories that reduce total traffic in the processor-memory interconnection.

It is not reasonable to have a detailed dependence on a number of processes on the order of thousands. The optimal number of processes depends on system hardware configuration and load, and neither number is a good basis for algorithm design.

Global parallelism, which places parallel processes at the top of the program hierarchy, is preferred to encapsulation, which confines parallel processes within individual program modules. An example was presented using the solution of block tridiagonal systems.

The NYU Ultra Computer: Its Architecture and its Applications

Malvin Kalos
NYU

The Ultracomputer project is aimed at the development of a MIMD machine comprised of a large number of small processors, the number ranging up to perhaps 8192. It will incorporate shared memory plus caches and will be organized around a message switched Omega network. Messages will be combined to handle "hot spots." Distributed control will incorporate dynamic load balancing and parallel queuing/dequeuing. Process creation will not be a critical section. This design supports a traditional programming style as well as coarse-grained data flow.

A shuffle-exchange network, topographically equivalent to an omega network, will be used for interconnection among processing elements and memories. This network follows an n log n growth pattern with the number of interconnected elements.

The fetch-and-add instruction is a universal coordination primitive. Given an address and an increment, it returns the contents of the addressed word in shared memory and adds the increment to the value stored there. This instruction is used for assigning work to processors without a centralized monitor. The execution of fetch-and-add is distributed through the network to avoid communications "deadlocks". The network switches have memory to maintain bandwidth for large systems and handle the combining of multiple fetch and add instructions.

Simulations of parallel programs using WASHCLOTH have been performed on a wide variety of applications, such as fluid dynamics, Monte Carlo, and molecular dynamics. Some performance evaluation/monitoring issues have been identified:

* network performance - queue use, blocking, combining
* cache performance
* process behavior - overheads, synchronization delays
* job/task throughput - multiprogramming scheduling

Among the conclusions of the Ultracomputer research (including network and cache simulations) are:

* distributed coordination is necessary
* effective caches can be designed
* programming need not be difficult
* a family of similar machines may be built, ranging from desktop to supercomputer

An 8-PE (68010) prototype with a UNIX based operating system has been built and can be accessed via the ARPAnet. IBM is building a many-processor prototype with significant architectural enhancements (RP3).

The Rediflow Simulator

Robert M. Keller
University of Utah

The name Rediflow is derived from reduction plus dataflow. Applications that have been coded and run include a relational database system, Prolog and LISP interpreters, matrix multiplication, tree searches, and convolution. Reduction is a fundamental problem-solving technique. It is also a medium-grain tasking model, including a full lambda-calculus. Processes are definable as tail-recursive reductions.

Rediflow integrates with reduction two forms of dataflow: streams and Kahn processes. A number of add-ons are available, such as "object-oriented" programming, indeterminate "flow" operators, open Von Neumann operations, logical variables, unification, and resource expressions. Among the motivations for this effort are scalability (to tens of thousands of processors), determinacy by default, completely distributed dynamic load balancing, and programming with minimal configuration sensitivity.

The Rediflow architectural model is developed around the "Xputer" as a replicable unit; this comprises a smart switch, a processor (such as a 32 bit microprocessor) and a one megabyte memory. Various switching networks are possible. Using the concept of load balancing by pressure gradient, it was asserted that under "steady-state" conditions, propagated pressure yields the minimum number of hops to an underutilized Xputer. The route to the latter is dynamically established.

Knobs and meters in the simulator provide a means by which various performance parameters can be monitored and manipulated.

The current simulator runs on VAX/Unix and the DEC-20. It comprises 10K lines of Pascal. It is slow and memory-limited. Plans for the future include:

* more efficient compiled code
* calibrated simulation and more interaction (with Lisp)
* emulation (on the Butterfly)
* small-scale prototype (by Sperry)

Evaluating the Performance of the PASM Reconfigurable SIMD/MIMD Machine

James T. Kuehn
H. J. Siegel
Leah H. Jamieson
Purdue University

PASM (Partitionable SIMD/MIMD) is a dynamically reconfigurable multimicroprocessor system being developed at Purdue University for research in large-scale parallel image and speech understanding applications. PASM consists of N Processing Elements (PEs) that communicate through a multistage cube-type network, Q Microcontrollers (MCs), each of which controls N/Q PEs, N/Q secondary storage devices, and a number of processors that operate in a distributed fashion for high-level task scheduling and memory management functions. A prototype of PASM with N=16 and Q=4 is currently being constructed.

An MC and its N/Q PEs (called an MC-group) can act either as an SIMD machine (with the MC acting as an SIMD control unit) or as an MIMD machine. Also, the MC-group can dynamically switch from one mode of parallelism to the other. MC-groups can be combined to form larger virtual machines of up to N PEs and the virtual machine size can be changed at run time. Since there are Q such independent MC-groups available, PASM can support multiple simultaneous users. The various virtual machines all function independently due to the interconnection network partitionability and distributed control. Because of the interconnection network reconfigurability, a variety of communication patterns such as mesh, ring, and pyramid are possible.

The reconfigurability of the PASM network, variable virtual machine size, and the dynamic switching between SIMD and MIMD modes make performance measurement and analysis difficult on a system-wide basis. Analysis techniques used to date include detailed simulation studies of SIMD and MIMD algorithms at the machine-language level, interconnection network performance simulations, and parallel algorithm complexity studies. Data collection hardware is being implemented in the prototype.

Another aspect of the project is the study of matching parallel algorithms to architectures. Algorithm programming constructs and architecture features are examined pairwise; for example:

local storage requirements vs. local memory size
local/global references vs. memory structure and access times
communication structure vs. interconnection network type

This matching is often aided by application-specific characteristics.

Further information about the PASM architecture, parallel image and speech algorithm performance studies, and studies of matching algorithms to architectures is available from the authors.

Experience with HEP

S. Lakshmivarahan
University of Oklahoma

Experimental evaluations were carried out on various classes of problems with various degrees of parallelism. Hockney and Jesshope present the following formulation:

$$t = r^{-1} [n + n_{1/2}]$$

"N-half", the vector length required to obtain half the maximum performance, is generally considered a good discriminant. This formulation purports to represent the relationship between speed of computation and degree of parallelism.

From experimental results it has been observed that computation speed does not decrease linearly with the degree of parallelism. In the situations studied on the HEP -- cascade sum, linear recurrence, and solving Ax=b where A is a tridiagonal matrix -- saturation was observed at about 12 to 16 processors. The factor "n-half" did not appear to be a particularly useful parameter; megaflops appeared to be preferable.

A Benchmark Comparison of Three Supercomputers: Fujitsu VP-200, Hitachi S810/20, and CRAY X-MP/2

Olaf Lubeck
LANL
Los Alamos, New Mexico

Performance measurement of supercomputers is strongly dependent on workload. Measurement of a supercomputer yields an entire spectrum of performance and a workload characterization identifies only a point on that spectrum that may vary in time.

The Fujitsu VP-200, Hitachi S810/20, and the Cray X-MP/2 were benchmarked, using codes that typify the Los Alamos workload. Benchmarking was approached at three levels:

1) Timing of elementary vector operations as a function of vector length, to obtain basic data about the architecture and compiler.

2) Timing of short characteristic codes, ranging from purely scalar to highly vectorized codes, up to 3000 lines of FORTRAN. This level includes algorithms such as matrix operations, linear algebra routines, and fast fourier transforms, as well as large code excerpts such as particle in cell, Monte Carlo, state equations, and hydrodynamics.

3) Timing of characteristic real codes that can contain up to 20,000 lines of FORTRAN. These codes were not used in this benchmark.

The benchmark set consisted of 10 programs derived from major codes run at Los Alamos. Most codes were between 400 and 3000 lines of FORTRAN. Most were originally developed on Cray machines. The amount of vectorization varies from 0 to 99 percent.

The overall conclusion was that the VP-200 is "comparable" to the American supercomputer. The VP-200 can be two to three times as fast as the X-MP/2 in vector mode for large vector lengths. On codes indicative of the Los Alamos workload, the two machines are comparable. The S810/20 was a factor of two slower than the other two machines, probably because its scalar performance and vector processor clock period are slower.

The Importance of Performance Analysis Tools in Highly Parallel Architectures

Creve Maples
Vitesse Electronics
Camarillo, California

The comparison of computers may presently be characterized as "chaotic." Better tools are needed for use within individual systems, let alone intercomparison of systems. Areas of concern include the following:

Hardware
- Communication structures
- Memory organization
- Control
- Contention

Software
- Problem decomposition
- Reproducibility
- Synchronization
- Sequential code
- Problem dispatching

The MIDAS concept is a parametrically controlled configuration. Arranged in a pyramid structure, a three-level system would have 137 processors, while a four-level system would have some 1200 processors. A modified hierarchical memory is employed, with some portions being local and some shared. Processors can be interconnected via memory. A variety of interconnection networks among processors can be established to achieve algorithmically-specialized processor arrays. This system has been used to perform alpha-beta tree searches in artificial intelligence. It has been programmed for the game of Othello, where it is feasible to carry out comprehensive look-ahead procedures. Different processors can be used for different functions. For example, in an 8-processor configuration, one processor might be dedicated to the processing of serial code, while 2 or 3 are used for load balancing and the others are used for communications.

The two main problems are implementing the algorithm and achieving load balancing. An important aspect in analyzing such a machine is to determine how the time is being spent. Analytical tools are necessary for isolating bottlenecks.

Analysis and Measurement of Scientific Applications Codes

Joanne L. Martin
IBM/Yorktown

The purpose and goals of this project are to:

* obtain a collection of reasonable problems in the engineering/scientific domain
* develop computational models of these problems
* develop models of significant architectures
* assess the relationship between the computational and architectural models
* answer the question: Is there an inherent pairing of the models?

The codes selected for analysis include VA3D, SCF, Molecular Dynamics, WAVE, Monte Carlo, TRAC, and others of interest to universities and customers. Architectures to be studied are:

* standard scalar
* super scalar
* vector
* parallel
  - scalar with attached processors (loosely-coupled)
  - shared-memory with multiple processors (tightly-coupled)
  - shared-memory with multiple vector processors
* dataflow

Initial test architectures are the Cray X-MP, the IBM 3084 and 3090, the LCAP (Loosely Coupled Array of Processors), and the RP3. Tools for measurement and analysis will feature:

* software
  - instruction analysis tools developed at LANL for use on the Cray architectures
  - PARAPHRASE from the University of Illinois
  - other tools and simulators, as available
* hardware
  - IBM systems performance monitors

Benchmark runs are carried out using INSTRMIX. This method has the following steps:

* Compile the source code and obtain an assembly listing.
* Using a preprocessor, insert counting macros before each instruction in the listing.
* Assemble the code.
* Run the postprocessor to produce formatted counts and summaries of instructions executed.

The runtime is magnified by a factor of about 10. Timings are made in separate runs. Preliminary measurements have been made for the WAVE and VA3D codes.

Nonparallelizable Algorithms

Ernst Mayr
Stanford University

Parallel complexity theory deals with:

* parallel machine models
  - P-RAM and its variants
  - fixed interconnection networks
  - VLSI

* parallel complexity classes
  - parallel time approximates sequential space
  - NC = problems solvable in polylog time using a polynomial number of processors

Problems in NC are said to be "efficiently solvable" in parallel. Problems "complete for P" (where P = polynomial time), under logspace reductions, are commonly considered non-parallelizable, since an NC-algorithm for any one of them would imply P = NC and, in particular, P is contained in polylogspace. Examples of P-complete problems include the circuit value problem, network flow problem, list scheduling, linear programming, and unification.

Some problems, with many solutions, have extremely simple, optimal sequential algorithms, e.g. "greedy graph traversal", the maximal independent set problem, and the first-fit bin-packing heuristic. However, to compute the same solutions as computed by the (standard) sequential algorithms is P-complete. Nonetheless, all of the above problems have NC solutions and these efficient parallel algorithms use completely different approaches. Very often, algorithms using the "greedy" paradigm turn out to be non-parallelizable (P-complete). Also, some P-complete problems permit efficient parallel approximation schemes.

In conclusion, certain problems permit no significant speed-up whatsoever, independent of the machine model, dataflow, functional programming language, etc. For other problems, the standard algorithms are inherently sequential, but there are very different, fast parallel algorithms, generally producing very different solutions. As in the sequential case (P vs. NP), approximation schemes sometimes provide fast, viable parallel solutions.

Performance Monitoring of the Research Parallel Processor Prototype (RP3)

Kevin P. McAuliffe
IBM/Yorktown

A key element in the RP3 design is the performance monitoring chip. The performance monitoring chip counts the occurrence of events in the PME and stores sample information. It is controlled via the I/O Support Processors (ISP) or the PME and may operate in either a transparent mode or PME mode. The chip can monitor a variety of events such as instructions complete, translated and non-translated requests, cache hits and misses, waiting time, response time, and busy time. Sample information includes the virtual addresses of the last instruction and the last data reference, the last opcode serviced, and time stamping of requests and responses. A variety of commands are available for controlling and reading data from the chip, and for setting and clearing bits of its status register. The combining network can be monitored in detail, both for network performance and algorithm performance.

Why NBS Should Set the Computer Classification Standards

Harry Nelson
Lawrence Livermore National Laboratory

The purpose of benchmarking is to attach numbers to products. This facilitates at least a partial ordering and is an aid to understanding. There are frequent requests for a single number to characterize a machine. Two such metrics are the Nominal Service Unit (NSU) and the Relative Computing Unit (RCU), which has superseded the NSU, being larger by a factor of 10,000.

The RCU can be used to characterize a diversity of computing requirements over an extended time span for a particular laboratory. A variety of supercomputers can be ordered by class, according to speed (in MFLOPS), and can be plotted using a scale of MFLOPS or RCU's. The plot shows peak and net MFLOPS, with the net value being based on the weighted harmonic mean of expected usage.

The use of kernels has considerable merit as a way of characterizing machines with respect to particular applications, as these can be developed quickly. However, it is not advisable to average the results of different kernels. A factor that is not evident is the skill necessary to achieve the rated performance.

In conclusion, the development of a standard metric for computer performance should be an NBS responsibility, because the individual laboratories are unable to do it, and it should not be left to the vendors.

Performance Considerations in HEP SISAL

Rodney R. Oldehoeft
Colorado State University

SISAL is a dynamic data flow language, a descendant of the VAL (static) programming language. It includes these features:

* applicative semantics
* infix forms, block structure, strong typing
* rich data structures
* streams for I/O and pipelining
* parallel and sequential loops
* enriched value types from loops

It does not provide explicit parallelism and synchronization, but it has specific provisions to preclude deadlock.

Cooperative efforts with other institutions include:

* DEC - a multi-cpu VAX, VMS parser, code generator, and interpreter-executed benchmarks

* LLNL - vector processors, UNIX interpreter, and tools for the intermediate form (IF1)

* University of Manchester - dataflow machine with an instruction set optimized for SISAL

* Colorado State University - Denelcor HEP, UNIX parser, an IF1 to IFPCC translator, and HEP/UPX run time support

Among areas of current study are: machine independent optimization, dynamic storage allocation, arrays, process management, and parallel loop execution.

Plans for the NSA Supercomputing Research Center

Paul Schneck
NSA

The NSA Supercomputing Research Center (SRC) is a division of the Institute for Defense Analyses. In addition to its advanced computing facilities, it will have a broadly based research staff. Its goals are to:

* advance the state-of-the-art in supercomputing
* develop and evaluate parallel processing systems
* develop parallel processing algorithms and systems for solving national security problems
* establish and maintain a secure center of excellence for supercomputing research

The Center will engage in algorithm design and analysis, development of systems and applications software, and exploration of experimental architectures. These studies will include the transformation of programs for parallel architectures and intercommunication within such machines. Research will focus on operating systems, programming languages and compilers, and performance measurement. A substantial portion of the Center and its activity will be unclassified in order to facilitate interaction with outside researchers.

The Center will be staffed by approximately 100 professionals including computer scientists, mathematicians and engineers, with about 70 support personnel. It will contain an operational supercomputer, together with various support computers and prototype equipment. Work stations will be extensively provided. The facility, to be completed in 1988, will be located at the Maryland Science and Technology Center, near Bowie, Maryland. Until then work will be performed at an interim facility of 63000 square feet in Lanham, MD.

The Supercomputing Research Center will maintain close working relationships with industry and the academic community. Through contracts, industry will provide basic computation equipment and research prototypes. The SRC will work with universities to explore supercomputer architectures. The expertise of professionals in industry and academia in theory and implementation of parallel processing supercomputer systems will be solicited. Unclassified research results will be disseminated through publications and conferences.

Parallel Workloads

Herb Schwetman
MCC Corporation
Austin, Texas

Workloads are defined as collections of programs (and data) to be executed on a system. The purpose of this project is to develop workloads for "novel" architectures.

The objectives of novel architectures are generally to obtain enhanced performance and to reduce the running times of individual programs. Representative workloads are needed for use in designing, evaluating, and testing new systems, to help evaluate the impact of architectures.

Possible sources for such workloads include:

* Existing programs. These are of limited value because they do not use the novel features.

* Automatic generators, such as a vectorizer or parallel-erizer. This requires automatic detection of constructs that can be mapped onto new features.

* Assembly languages for the new systems. These may be expensive and limited.

* Compilers with new data objects and (possibly) new operations.

There is a great deal of experience with the progression from problem statement to programming language to machine language, but it is very hard to go the other way. The solution proposed is the development of a programming environment in which all of the levels are available. Thus it would not be necessary to depend on the machine to recognize the overall structure of a program from the fine structure.

Programming and Instrumentation Environment for Parallel Processing

Zary Segall
Carnegie-Mellon University

The key goal of the Programming and Instrumentation Environment for Parallel Processing (PIE) is semiautomatic generation of performance efficient parallel programs. Three levels of support are provided:

* Modular Programming Metalanguage
  - provides support for the efficient manipulation of parallel modules, fast parallel access to shared data constructs, and "programming for observability" in a language independent fashion
  - allows programmers to code in their choice of languages
  - provides the ability to observe or monitor the execution

* Program Constructor
  - provides mechanisms and policies for combining and transforming both the development time (static) and the run-time (dynamic) information into meaningful program representations
  - based on using the relational model approach

* Implementation Assistant
  - predicts parallel program performance before extensive implementation investments
  - assists the user in choosing between "implementations," that is, specific ways of decomposing and controlling the parallel computation processes and data

The PIE system is developed to be used as the Programming Environment for the Supercomputer Workbench (SCW). SCW will perform a dual role. First, it will support the performance prediction and evaluation of alternative algorithms, implementations and architectures for supercomputers. Second, it will be a host providing the infrastructure and supporting the integration of other special purpose architectures. The SCW hardware has been built with the goal of high programmability in mind. Hence, the hardware provides for homogeneity, resource allocation transparency, synchronization "hot spots" avoidance and non-intrusive hardware instrumentation.

The current PIE system (PIE 4) is developed to run on a VAX 11/784 processor.

Performance Measurements for the Butterfly Parallel Processor

Robert H. Thomas
BBN Laboratories

The Butterfly Parallel Processor is:

* an MIMD machine
* a tightly coupled, shared memory machine
* expandable over a wide range of configurations
* a homogeneous multiprocessor

The processor nodes utilize MC68000 processors, an AMD2901-based coprocessor, EPROM and 1 MByte of memory. The Butterfly switch is a high performance shuffle exchange network used to interconnect the processor nodes.

Objectives of the benchmarking effort are to measure:

* The ability of various applications to effectively utilize multiple processors.
* Multiprocessor overhead.

The benchmarking technique is to:

* Measure the runtime on 1 through n processors.
* Compare the runtime of an optimized uniprocessor version of the application with the runtime of the multiprocessor version on a single processor.

The programming methodology addresses two principal concerns:

* Storage management, where the goal is to keep all memories equally busy to prevent the slowdown that occurs when many processors access a single memory. The approach is to uniformly distribute data across the memory of the machine.

* Processor management, where the goal is to avoid the slowdown that occurs when some processors are overloaded and others sit idle. The approach is to dynamically assign tasks to processors, and to decompose the problem into T tasks, where T >> P (the number of processors.

Applications are structured into "worker" procedures that perform the tasks, and "task generator" procedures that identify the "next" task for execution. A task generator procedure G takes as parameters a worker procedure W and a description of data. G calls W, specifying subsets of the data until the work is completed. Processors are used as they are available to execute the calls of W. Task generation is implemented in a distributed fashion, with each processor that performs tasks participating

in their generation.

Results on matrix multiplication, solution of simultaneous linear equations, and various image processing utilities reveal an almost linear speedup with up to 256 processors.

Performance Monitoring of the Research Parallel Processor Prototype (RP3)

Michael Tsao and Kevin McAuliffe
IBM/Yorktown

A key element in the RP3 design is the performance monitoring chip. The RP3 has cache and a shareable memory with each processing element, the shareable portion being dynamically partitionable between local and global usage. The performance monitoring chip counts the occurrence of events in the PME (define?) and stores sampled information. It is controlled via the 1/0 Support Processors (ISP) or the PME and may operate in either a transparent mode or PME mode. The chip can monitor a variety of events such as instructions complete, translated and non-translated requests, cache hits and misses, waiting, response, and busy time in cycles. Sample information includes virtual addresses of last instruction and data reference, last opcode serviced, and time stamping of requests and responses. A variety of commands are available for controlling and reading data from the chip. Included on the chip is a detailed status register. The combining network can be monitored in detail, both for network performance and algorithm performance.

Toward Architectures that Execute Algorithms Efficiently

Leonard Uhr
University of Wisconsin

Networks of 10^4 computers have been built and in the time frame of 1985 to 2001 it should be possible to build networks having 10^5 - 10^6 (or even 10^9?) computers. These are needed for important and difficult tasks such as perception and thinking, to develop real intelligent systems, rather than simply AI toy demos. Network topology can employ any possible graph, yet, to date, only the simplest have been built. These are almost all von Neumann machines, with only one node, pipelines, with a linear string, arrays and n-cubes. There are many other interesting candidates that are too big to simulate, but can be analyzed by other means.

Important goals in designing an architecture should be elegant simplicity and the ability to execute very large jobs in realtime. Some example tasks are the perception of moving objects, which may be distorted or obscured in unknown ways, and everyday problem solving, namely conversing and responding in a second or two. These tasks involve non-numeric as well as numeric processing, and will require massively parallel systems. This can be accomplished -- the brain is the existence proof!

Criteria for assessing good topologies include such factors as density, connectivity for fault-tolerance, and addressibility. Graphical representations, such as Petri nets and data flow graphs, can be useful for depicting information flow for both software and hardware.

Topologies have progressed from the single CPU case to complete graph structures such as the bus, ring, crossbar, and star, as well as the reconfiguring shuffle network. Point-to-point topologies of today include the tree, pipeline, n-cube, 2-D array, and linked cluster. In the future we may expect augmented trees, pyramids, augmented pyramids, Moore graphs and other dense graphs, and compounded clusters. A number of these possibilities were (briefly) examined.

Benchmarking at Sandia National Laboratory, Albuquerque

Walt Vandevender
Sandia National Laboratory

The benchmarking activity at Sandia has involved a number of machines, including VAX 11/780, VAX 11/785, VAX 8600, CDC 7600, Cray 1-S, several versions of the Cray X-MP, and the ELXSI 6400. The codes used are generally of two types:

* Batch benchmark codes
  - 10 codes, on the order of 150 - 600 lines
  - Examples include:
    SPEED - measures machine speed in Megaflops
    WHET - measures machine speed in Mips
    LIN6 - inner product of long vectors

* Benchmark user codes
  - 7 codes, on the order of 2.5K - 45K lines
  - Examples include:
    TIGER - Monte-carlo electron/photon transport code
    ADINA - 3-D finite element structures code

Comparisons of SPEED and WHET on the various machines were presented, and price/performance comparisons of the machines were computed. Price per megaflop ranged from $109K for a Cray X-MP/48 to $1.5 million for a VAX 11/780. Comparisons of the 10 batch benchmark codes on the three DEC machines and the ELXSI were also presented.

Several major codes are being multitasked. Speedup when using 2 to 5 tasks was presented for the 5 kernels of the SPEED code and for a 60x60 matrix problem (Ax =b). A set of timesharing benchmark scripts has also been developed.

Early Experience with the FLEX/32

Robert G. Voigt
NASA/ICASE
Hampton, Virginia

The FLEX/32 machine is an MIMD machine having a common bus which can accommodate up to ten local buses. Each local bus will support two boards with each board consisting of 4 megabytes of memory or a processor with a megabyte of memory. The memory can be allocated for local or shared use. Programmable arbitrators provide access to the bus. System V, concurrent C, and FORTRAN are available; concurrent FORTRAN will be added. System generation allows local/global configuration of memory.

The installation will be used for exploratory purposes and is designated a beta site. The FLEX/32 will be used to explore parallel solutions of PDEs, which will be formulated directly for parallel computation, rather than the retrofitting of "dusty decks." Performance evaluations will include studies of communication, synchronization, and memory configurations. The system will utilize new programming languages which are evolving for research in parallelism. Programming environments will include PISCES and BLAZE. PISCES is FORTRAN-UNIX based, presents levels of virtual machines, allows multiple granularities of parallelism, and affords portability. The new scientific language BLAZE is a functional programming language which allows multiple granularities of parallelism, has high level functions such as array operations, and supports portability.

USCOMM-NBS-DC

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See Instructions)	1. PUBLICATION OR REPORT NO. NBSIR-86/3395	2. Performing Organ. Report No.	3. Publication Date JULY 1986

4. TITLE AND SUBTITLE

Report on the National Bureau of Standards Workshop on Performance Evaluation of Parallel Computers

5. AUTHOR(S)

Sandra B. Salazar and Carl H. Smith

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The Systems Components Division of the Institute for Computer Sciences and Technology at the National Bureau of Standards is actively engaged in the development of techniques to measure and evaluate the performance of parallel computers. As a preliminary step, a workshop on performance evaluation was held in Gaithersburg, Maryland on June 5th and 6th, 1985. The goal of the workshop was to define the issues and problems involved in the development of benchmarks for large parallel computers. Thirty-six talks were given by representatives of government, industries, universities and research laboratories. The topics presented ranged from specific measurements of large parallel machines to the philosophical issues concerned with the development of universally applicable benchmarks. In addition to the formal talks, there were several lively discussions.

This document is a report on the workshop. A one-page synopsis of each presentation is included. The synopses were compiled from notes taken for the Systems Components Division and copies of the speakers' transparencies. Each contributor had the opportunity to review the synopsis of his or her presentation.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

benchmarks; computer measurement; parallel computing; performance evaluation

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

50

15. Price

$9.95