Hypercube Computer Research at the University of Michigan

J. P. HAYES*, R. JAIN*, W. R. MARTIN†, T. N. MUDGE*, L. R. SCOTT‡, K. G. SHIN* AND Q. F. STOUT*

Abstract Since 1984, the Advanced Computer Architecture Laboratory (ACAL) at the University of Michigan has been developing a research program concerned with the architecture and application of high-performance parallel computers. ACAL operates a 64-processor NCUBE/six hypercube acquired under a beta-site agreement with the manufacturer. This paper describes our early experiences with a hypercube research facility, and also surveys our current research activities in the following areas: performance evaluation, parallel algorithms, fault-tolerant computing, computer vision, and scientific computation.

1. Introduction

The concept of a hypercube computer can be traced to work in the early 1960's by Squire and Palais at the University of Michigan [Squire and Palais 1963]. They carried out a detailed paper design of a 4096-node (12-dimensional) hypercube in which, as noted by Thurber, "hardware considerations and hardware economy were secondary considerations to the ease of programming" [Thurber 1976]. The hardware requirements of the Squire-Palais machine were estimated to be 20 times those of the IBM Stretch, one of the largest and most complex computers then in existence. Although several large hypercube computers were subsequently proposed, notably CHOPP by Sullivan and his colleagues at Columbia University [Sullivan et al. 1977], such machines did not become practical until the 1980's when VLSI technology made it feasible to produce powerful single-chip 16/32-bit microprocessors, and RAM chips in the 1M-bit range. In 1985 three manufacturers, Intel, Ametek and NCUBE, introduced the first commercial hypercubes. Also in that year, the Advanced Computer Architecture Laboratory (ACAL) of the University of Michigan obtained a 64-processor NCUBE/six hypercube made by NCUBE Corp., and has since then served as the sole university-based beta site for this machine.

*Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109 †Department of Nuclear Engineering, University of Michigan, Ann Arbor Michigan 48109 ‡Departments of Computer Science and Mathematics, Pennsylvania State University, University Park, Pennsylvania 16802

The organization of ACAL's hypercube research facility is shown in Fig. 1. The hypercube nodes are based on a VAX-like 32-bit microprocessor with full IEEE-standard floating-point capability. Each node has 128K bytes of local RAM storage and 11 high-speed (1M byte/sec) input-output channels. Ten of these channels can be connected to neighboring nodes in the hypercube, thus allowing the NCUBE machine to be expanded to a 1024-node hypercube. An additional channel links each node to a host processor based on the Intel 80286, which manages the I/O system and the UNIX-like AXIS operating system. The programming languages supported are Fortran 77 and C, both with message-passing extensions, and NCUBE assembly language.

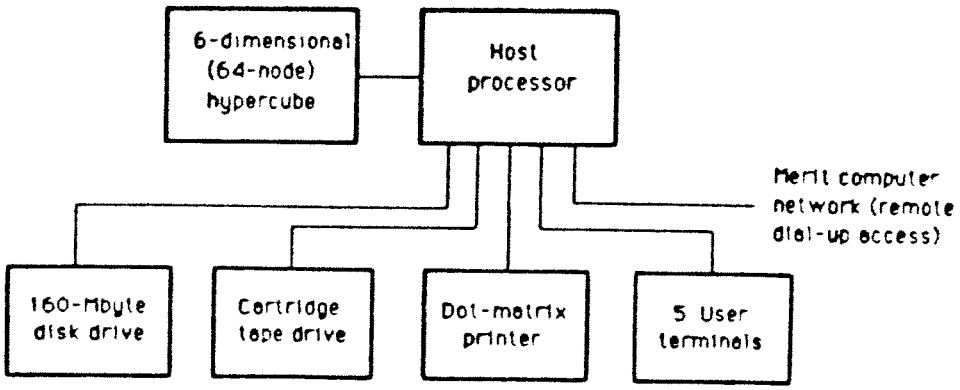

FIG. 1. The hypercube research facility at Michigan's Advanced Computer Architecture Laboratory as of Sept. 1986.

ACAL is concerned with research into the theory, design, and application of advanced computers, especially massively parallel architectures. It draws its membership primarily from the Computer Science Engineering division of the Electrical Engineering and Computer Science Department at the University of Michigan, but also has participants from the Mathematics and Nuclear Engineering Departments. Current research efforts are concentrated in the following areas: performance evaluation, parallel algorithms, fault-tolerant computing, computer vision, scientific computation. Recent and planned work in each of these areas is discussed in the remainder of this paper.

2. Performance Evaluation

Performance analyses attempt to quantify the behavior of a computer system—how many instructions can be executed per second, how quickly can data be retrieved from secondary storage, and so on. Our work at Michigan is pursuing three approaches to the performance analysis of distributed-memory machines. The first is collecting applications programs to be used as benchmarks. The second involves the creation of synthetic benchmarks, or synthetic workload generators (SWGs), by abstracting representative features from the benchmarks. Finally, the third involves taking the abstraction process further to develop analytical (usually stochastic) models of performance.

An important dichotomy in parallel machine architectures is that between machines with shared memory and those with distributed memory. The NCUBE/six and other hypercubes are representative of the class of parallel machines that have distributed local memories which can be accessed in normal memory access times. In contrast, accessing remote memory, i.e., memory associated with another CPU, takes an order of magnitude longer. Our work in performance evaluation is aimed at determining what limitations, if any, the local memory restriction places on the effectiveness of parallel machines, particularly, hypercubes. The initial phase of our work is concerned with developing a set of programs that can be used as benchmarks. These come from a wide variety of application areas, and draw heavily on the other research being performed at ACAL. Examples of functioning code that we have developed for the NCUBE/six include: synthetic benchmarks, specifically the Whetstone and Dhrystone programs [Hayes et al. 1986a,b]; matrix decomposition using standard Linpack routines; printed circuit board routing [Olukotun and Mudge 1986]; assorted image processing algorithms [Mudge and Abdel-Rahman 1987a,b]; problems in nuclear engineering using Monte Carlo simulation techniques [Martin et al. 1986]; sorting [Wagar 1986]; and machine learning.

Processor	Fortran Dhrystones/s	Fortran Whetstones/s*
NCUBE node processor at 8MHz	999	381,000
NCUBE node processor at 10MHz (est.)	1249	476,000
Intel 80286 (NCUBE host) at 8MHz with 80287 floating-point coprocessor	510	101,000
DEC VAX-11/780 with floating-point accelerator	741	426,000

*Double precision.

FIG. 2. Some performance measurements on the NCUBE node processor.

Figure 2 summarizes the results of some performance experiments, designed by Michigan student D. Winsor that compared the NCUBE node processor to two other CPU's with floating-point hardware: the Intel 80286/80287 (the NCUBE host processor served for this) and the DEC VAX-11/780 with a floating-point accelerator. The measurements were made with the NCUBE node and host processors running at 8 MHz. Extrapolated figures for the planned 10-MHz version of the NCUBE node processor are also given; they assume no wait states. Two widely used synthetic benchmark programs were employed in this study: the Dhrystone and the Whetstone codes. The Dhrystone benchmark is intended to represent typical system programming applications and contains no floating-point or vectorizable code. The original Dhrystone Ada code was translated into a Fortran 77 version with 32-bit integer arithmetic that attempted to preserve as much of the original program structure as possible. The Whetstone

benchmark, which aims to represent scientific programs with many floating-point operations, was used in a double-precision Fortran 77 version that closely resembled the original Algol code. The Dhrystone results in Fig. 2 are reported in "Dhrystones per second," each of which corresponds roughly to one hundred Fortran statements executed per second. The Whetstone figures represent the number of hypothetical Whetstone instructions executed per second.

The communication delay associated with neighbor-to-neighbor message passing in the NCUBE/six is indicated by Fig. 3, which is based on measurements made on an image-processing application program. With a 6 MHz clock, each message incurs an overhead of about 0.5 ms, primarily due to message buffer copying and the internode communication protocols employed by Vertex, the resident operating system kernel responsible for the store-and-forward message-passing function in the NCUBE. While this overhead is small compared to that of most other commercial hypercubes, there are applications which require faster message passing. We are developing a set of fast communication routines for these applications that bypass the normal NCUBE message-buffering steps [Mudge, Buzzard and Abdel-Rahman 1986].

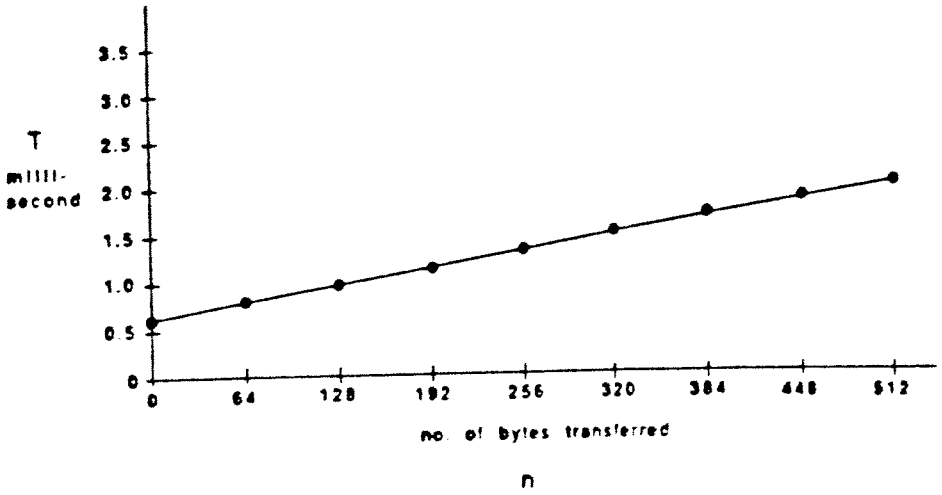

FIG. 3. Node-to-node communication delay T in sending an n-byte message.

3. Parallel Algorithms

The overall objective of this research is to design, analyze, and empirically measure the performance of a variety of parallel algorithms for solving nonumeric problems. It includes algorithms for sorting, routing, mapping graphs onto other graphs, optimization, computational geometry, and image processing. Approximation algorithms for NP-hard optimization problems are being developed with the general goal of obtaining efficient performance on distributed-memory medium-grained parallel machines, and the specific goal of efficient performance on hypercubes.

B. Wagar, a student at the University of Michigan has developed an internal sorting algorithm for the hypercube which has been measured to be significantly more efficient than Bitonic Sort [Wagar 1986]. For a hypercube of n processors each item is moved only $lg(n)$ times, instead of the approximately $lg^2(n)/2$ times required by Bitonic sort.

This sorting technique, which is called Hyperquicksort, is loosely based on Quicksort, and uses estimates for medians to reduce the number of times an individual data item must move. Like Quicksort, the partitioning may not be even, but with only a few hundred items per processor the observed average uneveness is quite small. We intend to push the analysis to prove that this is indeed the expected behavior, and to tune the algorithm for even better performance. We also intend to use Hyperquicksort to develop an external sort in which parallel disk I/O operations are used by the hypercube. The homogeneity of hypercubes encourages one to have a separate I/O channel into each node, but few algorithms have been developed which can utilize such a feature. The NCUBE hardware supports this, as do the FPS T-series machines to a lesser extent, and we expect that it will be widely supported in future hypercubes. For such a machine, efficient sort algorithms are likely to be important tools.

We have studied a variety of routing algorithms for specific message-passing tasks under a model of hypercubes in which the communication channels are the primary limiting factor and where each processor can use all of its channels simultaneously. This model is already approximately appropriate for the NCUBE series of machines, and other hypercube manufacturers are trying to develop nodes with this capability. We have shown optimal or nearly optimal algorithms for tasks such as broadcasting and transposing a matrix [Wagar and Stout 1986], but have not yet been able to implement them to determine the actual times. Slightly less efficient algorithms for these tasks have been developed at Yale [Saad and Schultz 1985]. One particularly interesting algorithm we have developed is a deterministic analogue of Valiant's randomized routing scheme for hypercubes [Valiant 1982]. Currently this requires that all processors know the routing permutation in advance, but we are also trying to find an efficient deterministic algorithm for the situation where the routing permutation is not known in advance.

The mapping of a task graph, where nodes represent modules and edges represent communication, onto a graph representing a parallel architecture, where nodes are processors and edges are communications links, is a well-studied problem in parallel processing. Typically one is trying to optimize some parameter, such as minimizing the maximum stretching of any edge, while satisfying some constraints, such as balancing the computational load of the processors. It is known how to map meshes, some trees, and pyramids onto hypercubes [Harary et al. 1986, Stout 1986], but little is known about mapping general graphs onto hypercubes. Stout and M. Livingston (a visiting scholar at Michigan) are working on this problem from several approaches. First they are trying to provide some bounds on the dimension needed to embed without stretching. For example it is proved in [Garey and Graham 1973] that if a bipartitite graph of n nodes can be embedded into some hypercube without stretching (in which case it is said to be *cubical*), then it can be embedded into a hypercube of dimension $n/2$ without stretching. We have extended this to show that if a graph of n vertices with minimum node degree 2 is cubical, then it can be embedded into a hypercube of dimension $2n/3$ without stretching, and further, this bound is the best possible.

Many computationally important problems in routing, scheduling, and packing are NP-hard optimization problems. Because they are NP-hard, there is an extensive literature on various serial approximation algorithms for such problems. There are also proposals to use special-purpose parallel computers for such problems [Hopfield and Tank 1986]. However, it is not clear how to blend such algorithms together to obtain ones suitable for medium-grained distributed-memory machines, and it seems that several approaches will be needed. In one project we use a 2-tier approach to do wire routing for printed circuit board design [Olukotun and Mudge 1986]. We assign a region of the circuit to each processor, and first use a high-level view which decides which regions each

wire would cross. Then we use a local algorithm within each processor to route all the wires in the processor's region, making sure that boundaries matched properly. Once the parallel high-level algorithm has finished, each processor only needs to perform local computations or communications with a neighbor. This approach, which is also used on serial computers, matches medium-grained machines quite well, and seems to be applicable to other problems such as stereo matching of images.

A different approach is being used by a student R. Tanese in another research project. She is examining the neural network model, which has been suggested for optimization problems such as the traveling salesperson problem [Hopfield and Tank 1986]. This model sets up a "neural network" representing an instance of the problem, where some of the synaptic strengths in the network are somewhat randomly chosen, as are the initial neural activation levels. Then the model is iterated to produce a stable situation, which represents a solution which one hopes is legal and nearly optimal. To improve the solution, some of the strengths and/or activation levels are changed and the model is rerun.

4. Fault Tolerance

Fault tolerance, which is the ability to operate reliably in the presence of hardware of software failures, is a key requirement of high-performance computer systems in such areas as vehicle control and medical diagnosis. Its achievement requires comprehensive and fast testing of the system, detection and containment of error propagation, and reconfiguration to recover a fault-free operating condition. We have been conducting research for many years in the areas of fault modeling and test generation [Bhattacharya and Hayes 1985], error handling and fault tolerance models and techniques [Shin and Lee 1984], and recovery techniques for multiprocessors and distributed systems [Yanney and Hayes 1986, Lee and Shin 1984, Krishna and Shin 1986]. We are presently conducting an experimental evaluation [Woodbury and Shin 1986] of the error-handling characteristics of the FTMP and SIFT fault-tolerant multiprocessors at NASA's Langley Research Center. We have begun to study error detection and recovery in massively parallel distributed-memory systems using ACAL's NCUBE six system. We are also experimenting with the design of test generation and fault simulation programs for execution on both hypercubes and vector processors (the Cray-XMP).

The fault coverage that is achievable by practical testing schemes is severely limited by the computational cost of generating the test patterns; this cost rapidly increases with circuit size. Dramatic improvement in test generation would be possible if many test patterns could be generated in parallel; almost no work has been done to date on such techniques, however. We are investigating the problem of parallel test-pattern generation for very complex digital systems. The major goal of this effort is to obtain extremely high fault coverage i.e., to maximize the percentage of faults that are detected or isolated. A Ph.D. student D. Bhattacharya is developing a new hierarchical approach to this problem that can analyze faults and generate test patterns for them at several different levels of complexity, such as the gate and register levels [Bhattacharya and Hayes 1985]. Thus, instead of dealing with all faults at the conventional single-line or bit level, this methodology can manipulate vectors of lines (buses) or bits so that many faults can be handled in parallel during test-pattern generation. This approach can reduce test generation time, while allowing up to 100 percent of the traditional stuck at-0/1 faults to be detected.

When an error is detected in a multiprocessor system, the source of this error must be quickly identified so as to correctly reconfigure the system and recover from the error successfully. We are attempting to establish an experimentally validated model for the

study of error propagation in multiprocessors, including hypercubes, and to use this model for locating faults for a given detection mechanism. The basic component of the model is a unit with a single input and a single output. The error propagation property of the unit is characterized by a triplet (k,L,T), where k represents the pass rate, L the error latency, and T the error delay. The pass rate is the probability that an error in input will eventually induce an error in output. The error latency is measured from the time a fault occurred within the unit till an error was seen in output. The error delay is the time for an error to propagate from input to output within the unit. L and T are both random variables with distribution functions $F_L(\cdot)$ and $F_T(\cdot)$. The unit considered can represent any part of the system, i.e., it is not restricted to represent only processors, memories. It is useful to decompose the whole system into subsystems and model each subsystem as a unit. We are developing rules to combine two units into one larger unit so that once the properties of all subsystems are known, the property of the combined system can be derived. Due to the large number of units in a multiprocessor system, this combination often requires excessive computation. We have developed a parallel algorithm for the NCUBE six to calculate F_L and F_T for each of n units which interact with one another in an arbitrary way.

Since faults occur randomly and infrequently, it is difficult to study the behavior of faults and errors in a multiprocessor system without an artificial mechanism of injecting faults. Based on our extensive experience in using the fault injector for the FTMP, we are investigating the design of an improved fault injector for a large-scale hypercube multiprocessor. It is to be flexible enough to accommodate the injection of various faults, e.g., permanent faults, transient faults with varying active durations, intermittent faults, and malicious faults. We plan to develop auxiliary software to collect and dump data into a secure device. This data will be used in our parallel investigation of error propagation modeling, system diagnosis, and error containment. As a starting point, we have designed a prototype software fault injector for the NCUBE six at ACAL.

5. Computer Vision

This research is concerned with a variety of computation-intensive problems related to machine vision, image understanding, and their applications to such areas as sensor-based robot control. A major goal is to develop computer vision algorithms suitable for hypercube architectures in particular, and massively parallel architectures, in general. We are pursuing a new qualitative dynamic approach to image understanding which is based on exploiting redundant information. We wish to compare the performance of distributed-memory versus shared-memory multiprocessors for problems that involve reasoning about images. We are applying our results to problems in real-time control of robots with visual and other sensors as input.

The computational requirements of computer vision systems are extremely demanding, and are well suited to parallel processing. At the signal processing end, the number-crunching requirements may exceed several hundred MIPs (integer arithmetic), while at the cognitive end, very fast processing is also required. We have an extensive research effort at Michigan on dynamic vision [Jain 1984, Sethi and Jain 1986], range image understanding [Besl and Jain 1986] object recognition [Knoll and Jain 1986, Turney et al. 1985], and computer architecture for vision [Agrawal and Jain 1982, Miller and Stout 1985, Mudge and Abdel-Rahman 1987a,b]. We are presently addressing hypercube algorithm design and architectural issues related to some of the more computation-intensive aspects of computer vision.

Fast recognition of objects is one of the major goals of machine vision systems. We are studying object recognition using both range and intensity data. A major long term

goal of this effort is to develop techniques that use range information for object recognition and navigation. Range images contain explicit information about surfaces. This explicit information facilitates recognition and location tasks in many applications. Our emphasis in range image understanding is on finding robust symbolic surface descriptors that will be independent of viewpoint. We are developing techniques to characterize surfaces in range images with these symbolic descriptors. We have designed two new methods to recognize objects: the feature-indexed hypotheses method [Knoll and Jain 1986]; and the saliency-based method [Turney et al. 1985]. The first method breaks the recognition process into two phases: hypotheses generation and hypothesis verification. By using features that occur more than once in the possible object set, the number of features in the search can be greatly reduced. This method also has the advantage that unique features, which are difficult or impossible to find if the object set contains many similar objects, are not required.

We view image understanding as a dynamic process, which allows us to cope with the error-filled visual world by exploiting the availability of redundant information in an image sequence. We are currently developing a qualitative approach to this aspect of computer vision. This approach uses relative information available in a sequence to infer the relationships among objects in a scene. In dynamic vision we are addressing the following issues: segmentation, image flow, motion stereo, trajectories, and architecture for dynamic vision. The need to deal with sequences of images makes dynamic scene analysis a strong candidate for parallel processing using hypercubes.

Our past work has shown how many low-level computer vision algorithms (e.g. filtering) and mid-level algorithms can be redesigned to take advantage of hypercube architectures [Miller and Stout 1986, Mudge and Abdel-Rahman 1987b]. We have successfully implemented on the NCUBE/six an algorithm for reconstructing in three dimensions the submicron surface topography of an integrated circuit [Kayaalp and Jain 1986]. Work is also under way to parallelize an existing solder-joint inspection program [Besl, Delp and Jain 1985] for the NCUBE. The inspection technique employed is a statistical pattern analysis method that uses objective dimensionality reduction to select inspection features. It requires us to design and write parallel algorithms for extracting various classes of features from an image. As well as being important to our particular inspection techniques, many of these algorithms are of general use in computer vision. Two other applications being considered are a parallel version of our bin-of-parts algorithm [Mudge and Abdel-Rahman 1983, Turney, et al. 1985], and the development of parallel programs to support our work in dynamic scene analysis.

6. Scientific Computation

This section describes several research projects concerned with the algorithm design and software implementation for scientific computing applications using both hypercubes and conventional vector processors. These projects embrace some of the research issues in areas of scientific computation where supercomputers are most used. The major topics of the current research effort at Michigan are reactor plant simulation, Monte Carlo transport algorithms, logic circuit simulation, and distributed data structures. This work also aims at obtaining new insights into the important question of the relative efficiency of distributed-memory and shared-memory machines for large-scale scientific computations.

We are investigating the development and implementation of a nuclear reactor plant simulation model on hypercube architectures. The goal is to obtain a fast-running (faster than real time), reasonably accurate reactor plant simulation model that can execute satisfactorily on a computer other than an expensive conventional supercomputer.

The cost/performance characteristics of massively parallel architectures such as the hypercube makes them an attractive candidate for such a simulation. The intent is to have an economical and reliable reactor plant model that could be used within an operating plant, perhaps as a standalone plant simulator or as a component within a larger expert system.

The principal research task here is to partition the simulation algorithm across the processors of the hypercubes. Thus, one can assign the reactor to one or more processors or clusters of processors, the steam generator(s) to another cluster, the pressurizer to another cluster, etc., and let each solve the pertinent equations for its specific component. Communication between clusters of processors is necessary due to the flow of the reactor coolant (density, enthalpy, pressure, velocity, etc.) and is being accomplished via message passing. The partitioning of the component models within each cluster of processors requires some new work. Since we have had substantial experience over the past 10 years developing reactor component and plant models [Feng, Lee, and Martin 1981; Baggoura and Martin 1983], this does not pose any major difficulties.

We are conducting research on the implementation of a photon transport Monte Carlo algorithm on the requested parallel processors. As opposed to other types of Monte Carlo methods, particle transport Monte Carlo is characterized by a considerable amount of floating-point arithmetic and is probably one of the most computation-intensive methods in scientific computation. This work is a natural extension of our previous successful efforts to develop vectorized Monte Carlo algorithms for the CDC Cyber-205 vector supercomputer [Brown and Martin 1985], the Cray-XMP and Cray-2 supercomputers [Martin et al. 1986] and the IBM 3090/200 and /400 supercomputers, [Wan and Martin 1986]. To allow meaningful comparisons with the conventional algorithm, a conventional Monte Carlo code has also been developed and successfully benchmarked against a production-level photon transport code from Lawrence Livermore National Laboratory. At this time we have a series of realistic demonstration codes for Monte Carlo photon transport on vector supercomputers and parallel/vector supercomputers, as well as conventional sequential computers.

We have completed a number of preliminary experiments running a parallel Monte Carlo code for photon transport on the ACAL NCUBE/six [Martin, Wan and Mudge 1986]. As noted in [Martin et al. 1986], two alternative approaches were taken to develop parallelized algorithms for the NCUBE/six which made use of this change. Representative results will be given for one approach - replication of the problem on each of the processors. In this case, the entire code is replicated on each processor and the host processor reads in the input data, sends messages to each node describing the problem and giving the random seed, and receives the results from each node when it is done. Since each node receives a different random seed, it is possible to combine the results a posterior to produce a result which is equivalent to one large simulation. This is one great advantage of Monte Carlo, and this implementation simulates how a user might combine several smaller Monte Carlo calculations to achieve a result with better statistics. It should be noted that this results in a problem size that grows linearly with the number of processors. Figure 4 summarizes the results of simulations with a 49×40 mesh (1960 zones) with approximately 2700 photons per node. As can be seen, the performance is nearly linear with the number of nodes, as might be expected for this approach. Since the total elapsed time for N Monte Carlo calculations (with different random seeds) distributed to N nodes is a constant (approximately 205 s), it is clear that the maximum speedup is being observed, indicating that hypercubes are extremely well-suited to this application. The last column of Fig. 4 lists the number of microseconds to simulate (track) a single photon, a commonly-used performance measure

in this area. Comparable figures obtained for the optimal scalar code (2× faster) on conventional vector processors are 35 μsec/track (Cray-XMP/48) and 40 μsec/track (IBM 3090 and Fujitsu VP-200).

No. of nodes N	Processing time T_S	Elapsed time T_E	μsec/track
1	195	205	4920
2	195	204	2448
4	195	204	1224
8	195	205	615
16	195	205	308
32	195	204	158
64	195	205	78

FIG. 4. Performance of Monte Carlo photon transport program on the NCUBE/six.

We are also examining ways to automate the programming of algorithms that utilize distributed data structures in scientific computation [Scott, Boyle and Bagheri 1986]. This research involves introduction of language extensions to Fortran that allow code on one processor to access variables explicitly (by name only) that are stored in another processor. In our implementations to date, code written with these extensions is then converted into appropriate message-passing code via a preprocessor. (Implementations have been done for both the NCUBE and Intel hypercubes). Not only does this approach free the programmer from having to write the message-passing code each time, it assures that it will be done correctly. So far, experiments have been carried out for simple iterative and direct methods for solving linear equations, which is the most computationally intensive part of many scientific computation codes. More complex algorithms are currently being coded in the extended language and tested on the NCUBE/six.

References

(1) B. BAGGOURA and W. R. MARTIN, *Transient Analysis of the TMI-2 Pressurizer System*, Nuclear Technology, vol. 62, 1983, p. 159.

(2) P. BESL, E. DELP and R. JAIN, *Automatic Visual Solder Joint Inspection*, IEEE J. on Robotics and Automation, vol. 1, no. 1, 1985, p. 42-56.

(3) P. BESL and R. JAIN, *Invariant Surface Characteristics for 3-D Object Recognition in Depth Maps*, Computer Vision, Graphics and Image Processing, vol. 33, 1986, pp. 33-80.

(4) D. BHATTACHARYA and J. P. HAYES, *High-level Test Generation Using Bus Faults*, Proc. 15th Fault-Tolerant Computing Symp., June 1985, pp. 65-71.

(5) F.B. BROWN and W.R. MARTIN, *Monte Carlo Methods on Vector Computers*, Prog. in Nuclear Energy, vol. 14, 1985, p. 269.

(6) M.-S. CHEN and K.G. SHIN, *Embedding of Interacting Task Modules Into a Hypercube Multiprocessor*, submitted to SIAM J. on Computer, to appear Sept. 1986.

(7) M.-S. CHEN and K.G. SHIN, *Determination of a Minimal Subcube for Interacting Task Modules*, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986 (these Proceedings).

(8) Y.C. FENG, J.C. LEE and W.R. MARTIN, *Nonequilibrium Transient Two-phase Flow Modeling and Analysis*, Trans. Am. Nucl. Soc., vol. 39, 1981, p. 505.

(9) M.R. GAREY and R.L. GRAHAM, *On Cubical Graphs*, J. Combin. Theory A, 18, 1973, pp. 263-267.

(10) F. HARARY, J.P. HAYES and P. WU, *A Survey of the Theory of Cube Graphs*, Sept. 1986, submitted for publication.

(11) J.P. HAYES, T.N. MUDGE, Q.F. STOUT, S. COLLEY and J. PALMER, *The Architecture of a Hypercube Supercomputer*, Proc. 1986 Intl. Conf. on Parallel Processing, Aug. 1986a, pp. 653-660.

(12) J.P. HAYES, T.N. MUDGE, Q.F. STOUT, S. COLLEY and J. PALMER, *A Microprocessor-Based Hypercube Supercomputer*, IEEE Micro, vol. 6, no. 5, Oct. 1986b, pp. 6-17.

(13) J.J. HOPFIELD and D.W. TANK, *Computing With Neural Circuits: a Model*, Science, vol. 233, 1986, pp. 625-633.

(14) R. JAIN, *Segmentation of Frame Sequences Obtained By a Moving Observer*, IEEE Trans. PAMI, Sept. 1984, pp. 624-629.

(15) A.I. KAYAALP and R. JAIN, *The Parallel Implementation of an Algorithm for 3-D Reconstruction of IC Pattern Topography Using SEM Stereo on the NCUBE Machine*, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986 (these Proceedings).

(16) N. KHAN and R. JAIN, *Uncertainty Management in a Distributed Base System*, Proc. Intl. Joint Conf. on Artificial Intelligence, Los Angeles, Aug. 1985, pp. 318-320.

(17) T.F. KNOLL and R. JAIN, *Recognizing Partially Visible Objects Using Feature Indexed Hypothesis*, IEEE J. Robotics and Automation, vol. 2, 1986, pp. 3-13.

(18) C.M. KRISHNA and K.G. SHIN, *On Scheduling Tasks with a Quick Recovery from Failure*, IEEE Trans. Computers, vol. C-35, May 1986, pp. 448-455.

(19) Y.H. LEE and K.G. SHIN, *Design and Evaluation of a Fault-tolerant Multiprocessor Using Hardware Recovery Blocks*, IEEE Trans. Computers, vol. C-33, Feb. 1984, pp. 113-124.

(20) W.R. MARTIN et al., *Monte Carlo Photon Transport on a Vector Supercomputer*, IBM Jour. of Res. and Dev., March 1986a.

(21) W.R. MARTIN, D. POLAND, T.C. WAN, T.N. MUDGE and T.S. ABDEL-RAHMAN, *Monte Carlo Photon Transport on the NCUBE*, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986b (these Proceedings).

(22) R. MILLER and Q.F. STOUT, *Geometric Algorithms for Digitized Pictures on a Mesh-connected Computer*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-7, 1985, pp. 216-228.

(23) R. MILLER and Q.F. STOUT, *Data Movement Operations for Mesh-of-trees and Hypercube Computers*, submitted for publication, 1986

(24) T.N. MUDGE and T.S. ABDEL-RAHMAN, *Case Study of a Program for the Recognition of Occluded Parts*, Proc. 2nd Annual IEEE Workshop on Computer Architecture for Pattern Analysis and Image Data Base Mangement, Pasadena, CA, Oct. 1983, pp. 56-60.

(25) T.N. MUDGE, *The Next Generation of Hypercube Computers*, Proc. of ARO Workshop on Future Directions in Computer Architecture and Software, May 1986

(26) T.N. MUDGE, G.D. BUZZARD and T.S. ABDEL-RAHMAN, *A High-Performance Operating System for the NCUBE*, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986 (these Proceedings)

(27) T.N. MUDGE, and T.S. ABDEL-RAHMAN, *Architectures for Robot Vision*, Specialized Computer Architectures for Robotics and Automation, Ed. J. Graham, Publ. Gordon and Breach, Inc., 1987a. (to appear)

(28) T.N. MUDGE and T.S. ABDEL-RAHMAN, *Vision Algorithms for Hypercube Machines*, Journal of Parallel and Distributed Computer, 1987b, (to appear)

(29) T.N. MUDGE, J.P. HAYES, G.D. BUZZARD and D.C. WINSOR, *Analysis of Multiple bus Interconnection Networks*, Journal of Parallel and Distributed Computing, 1987, (to appear)

(30) O.A. OLUKOTUN, and T.N. MUDGE, *Parallel Routing on a Hypercube Computer*, submitted to the 24th Design Automation Conf., 1987

(31) Y. SAAD and M.H. SCHULTZ, *Data Communication in Hypercubes*, Tech. Rept., YALU/DCS/RR-428 Dept. of Computer Science, Yale U., 1985

(32) L.R. SCOTT, J.M. BOYLE and B. BAGHERI, *Distributed Data Structures* for Scientific Computation, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986 (these Proceedings)

(33) I.K. SETHI and R. JAIN, *Finding Trajectories of Feature Points in a Monocular Image Sequence*, IEEE Trans. PAMI, Nov. 1986.

(34) K.G. SHIN and Y.-H. LEE, *Error Detection Process: Model, Design, and Its Impact on Computer Performance*, IEEE Trans. Computers, vol. C-33, June 1984, pp. 529-540

(35) K.G. SHIN and Y.-H. LEE, *Evaluation of Recovery Blocks Used for Cooperating Processes*, IEEE Trans. Software Engineering, vol. SE-10, Nov. 1985, pp. 692-700

(36) J.S. SQUIRE and S.M. PALAIS, *Programming and Design Considerations for a Highly Parallel Computer*, AFIPS Conf. Proc., vol. 23, 1963 SJCC, pp. 395-400

(37) Q.F. STOUT and B. WAGAR, *Passing Messages in Link-bound Hypercubes*, presented at Second Conf. on Hypercube Multiprocessors, Knoxville, Tenn., Sept./Oct. 1986 (these Proceedings).

(38) Q.F. STOUT, *Hypercubes and Pyramids*, in Pyramidal Systems for Image Processing and Computer vision, V. Cantoni and S. Levialdi, eds., NATA ASI Series ARW, Springer-Verlag, 1986, to appear

(39) H. SULLIVAN and T.R. BASHKOW, *A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I*, Proc. 4th Ann. Symp. on Computer Architecture, 1977, pp. 105-117

(40) K.J. THURBER, *Large Scale Computer Architecture*, Hayden, Rochelle Park, NJ., 1976

(41) J L TURNEY, T N MUDGE and R A VOLZ, *Recognizing Partially Occluded Parts*, IEEE Trans on Pattern Analysis and Machine Intelligence, vol PAMI-7, July 1985, pp 410-421

(42) L G VALIANT, *A Scheme for Parallel Communication*, SIAM J Computing, vol 11, May 1982, pp 350-361

(43) B WAGAR, *Hyperquicksort—a Fast Sorting Algorithm for Hypercubes*, presented at Second Conf on Hypercube Multiprocessors, Knoxville, Tenn, Sept /Oct 1986 (these Proceedings)

(44) T C WAN and W R MARTIN, *Parallel Algorithms for Photon Transport Monte Carlo*, accepted for presentation at the Winter meeting of the American Nuclear society, Washington, DC, Nov 1986

(45) M H WOODBURY and K G SHIN, *Performance Modeling and Measurement of Real-time Multiprocessors with Time-shared Buses*, IEEE Trans on Computers, 1986 (to appear)

(46) R YANNEY and J P HAYES, *Distributed Recovery in Fault-tolerant Multiprocessor Networks*, IEEE Trans on Computers, vol C-35, Oct 1986, pp 871-879