LA-UR -86-4356

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE. EXPERIENCES AND RESULTS MULTITASKING A HYDRODYNAMICS CODE ON GLOBAL AND LOCAL MEMORY MACHINES

LA-UR--86-4356

DE87 003759

AUTHOR(S) David A. Mandell (X-7)

SUBMITTED TO. 1987 International Conference on
Parallel Processing
August 17-21, 1987
St. Charles, Illinois

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

EXPERIENCES AND RESULTS MULTITASKING A HYDRODYNAMICS CODE ON GLOBAL AND LOCAL MEMORY MACHINES

by

David Mandell
Computational Physics Group
Applied Theoretical Physics Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

A one-dimensional, time-dependent Lagrangian hydrodynamics code using a Godunov solution method has been multitasked for the Cray X-MP/48, the Intel iPSC hypercube, the Alliant FX series and the IBM RP3 computers. Actual multitasking results have been obtained for the Cray, Intel and Alliant computers and simulated results were obtained for the Cray and RP3 machines. The differences in the methods required to multitask on each of the machines is discussed. Results are presented for a sample problem involving a shock wave moving down a channel. Comparisons are made between theoretical speedups, predicted by Amdahl's law, and the actual speedups obtained. The problems of debugging on the different machines are also described.

I. INTRODUCTION

In order to understand the methods required in multitasking on different parallel processors, a one-dimensional, time-dependent Lagrangian code was multitasked for a Cray X-MP/48, an Intel iPSC hypercube, an Alliant FX/6 and an IBM RP3.

The Cray X-MP/48 has four processors sharing a common memory and the Intel hypercode has up to 128 nodes (processors), each similar to an IBM PC/AT, with local memory. Multitasking on the two machines is therefore substantially different. One purpose of this work was to understand these differences. The Cray multitasking was done using the Los Alamos Multitasking Library and simulator [1]. The Los Alamos Library is a superset of the Cray multitasking library [2]. References 3 and 4 describe the use of the Intel iPSC hypercube and the Fortran implemented on the machine.

The Alliant FX/series of vector, parallel computers [5] allow parallel processing with very little effort on the part of the programmer. In this case only a one line change was required to the UNIX* serial version of the code.

The RP3 is currently being designed and built at the IBM Thomas J. Watson Research Center. A general description of the machine is given in Ref. 6 and the simulator, EPEX, is discussed in Ref. 7. Initially, the RP3 will have 64 processors with the unique feature that the fraction of total memory used as local memory can be specified at run time. The coupled hyperbolic partial differential equations for the conservation of mass, momentum and energy are described by Richtmyer and Morton [8] and summarized in the next section.

The Fortran changes needed in order to multitask on each machine are then described. (The original code was a serial code written for the Cray.) Finally, the results are presented and discussed.

BASIC EQUATIONS

The one-dimensional hyperbolic equations involve three equations for four unknowns. The fourth unknown is found from the equation-of-state of the material, which is an ideal gas in this work [8]. The equations are

$$\frac{\partial U}{\partial t} + \frac{\partial}{\partial x} F(U) = 0$$

where

$$U = \begin{bmatrix} v \\ u \\ E \end{bmatrix} \text{ and } F(U) = V_0 \begin{bmatrix} -u \\ p \\ pu \end{bmatrix}$$

*UNIX is an AT&T trademark

V = specific volume,

V_0 = initial specific volume; given,

u = velocity,

E = total energy,

p = pressure,

t = time, and

x = position

V, u and E are found from the above equations and then p is found from the ideal gas equation

$$p = (\gamma - 1)\, I/V$$

where

$I = E - \frac{1}{2} u^2$ and γ, the ratio of specific heats, is 1.4 in this case.

FINITE-DIFFERENCE EQUATIONS

The dependent variables (u, V, p, and E) are cell-centered and the Riemann pressures and velocities (P12, and W12), obtained from the Godunov solution [9], are at the cell edges. For the jth cell, where j is at the cell center,

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{\Delta r} \left(F_{j+1/2}^{n+1/2} - F_{j-1/2}^{n+1/2}\right)$$

(Note, Ref. 8 has a sign error in this equation).

$$F_{j+1/2} = V_0 \begin{bmatrix} -W12_{j+1/2} \\ P12_{j+1/2} \\ (P12 \cdot W12)_{j+1/2} \end{bmatrix}$$

Where Δr is the initial distance between cell edges and P12 and W12 are found by using the approximate Riemann solver given in Ref. 9.

CODE CHANGES

Changes from the Cray serial code, in order to mulitask on each machine, are described in this section.

CRAY X-MP/48

The code was multitasked by dividing the calculations of the serial computational cells into a specified number of tasks in a self-scheduling manner. That is, the number of computational cells per task is determined during execution. For example, with 100 computational cells which were used in the base case runs for this work, and four processors, 25 cells are scheduled at a time.

Only a small number of changes were required to the sequential code in order to implement the Los Alamos Multitasking Library constructs. Tasks are initiated at two points in the program for each cycle. First tasks are started to calculate the new Riemann pressures and velocities at each cell interface from the old cell-centered variables. Since these calculations cross task boundaries, they have to be completed before the second set of tasks are started, which calculate the new cell-centered variables.

The only significant debugging problem that occurred in creating the Cray multitasking code involved two Fortran lines. In the serial code two local variables were set only for the processor doing the first computational cells. The processor(s) doing the rest of the cells had undefined values for these variables and the calculation became unstable. This trivial error in the multitasked code, which runs correctly on one processor, took a long time to find and illustrates the problems of debugging even a small multitasked code.

iPSC Hypercube

Two programs are needed for the cube, a host program that controls the job and does the I/O and a node program which is loaded on each node of the cube that is being used in the current job. The node program does the execution for a particular part of the calculation, similar to the task on the Cray.

Since the cube is a local memory machine, in contrast to the Cray's global memory, messages must be passed between nodes during each time cycle. Each node needs to get its boundary conditions from adjacent nodes.

Fortran changes to the serial code were required in order for the code to compile on the cube. Namelist is not allowed in the cube. Also slashes and double quotes are not acceptable in format statements. In order to execute, the 1.0E-50s in a Cray code subroutine had to be changed since they caused underflows on the cube. Underflow errors occurred at the point where the solutions go to zero so checks had to be inserted and the values of the variables were not allowed to be smaller than 10^{-30}. The Cray conditional vector merge subroutine was replaced by a Fortran equivalent. These changes did not effect the results.

Some problems occurred in debugging because the hardware and software were new. It was necessary to write messages to a log file in order to trace the code flow and determine the location of execution errors.

ALLIANT FX/6

The starting point for multitasking on the Alliant was a serial version of the code running on a VAX/780 under UNIX. Only two minor changes were required. Double quotes in format statements had to be changed to single quotes and a single line directive had to be added to tell the compiler that a subroutine call in a DO loop did not have a dependency. The entire effort to obtain a correct multitasked code on the Alliant was trivial.

RP3

The serial code discussed above was multitasked for the RP3 simulator by Dr. Frederica Darema-Rogers of the IBM Thomas J. Watson Research Center. From the code listings, the changes appear very straightforward.

RESULTS AND DISCUSSION

The results of this work are a comparison between the machines of the following:

1) The difficulty of converting the serial code to the multitasking code;
2) global memory vs local memory;
3) debugging problems; and
4) speedups and efficiency.

The speedup, S, is defined as

$$S = \frac{\text{time for nonparallel code to run (1 processor)}}{\text{time for parllel code to run (NP processor)}}$$

It should be noted that the numerator is not the one processor multitasked code time. The overhead due to multitasking, which may be substantial in some cases, has to be considered as a penalty in the speedup; and, thus the serial time should be used in the numerator. The efficiency is the speedup divided by the number of processors

$$e = S/NP$$

The theoretical speedup can be obtained from Amdahl's law if the fraction of the code that can be made parallel is obtained from a serial run. This can be used to determine if it is worthwhile to multitask a given code. Amdahl's law is

$$S = \frac{1}{1 - P + P/NP}$$

where

p is the fraction parallel
NP is the number of processors
and S is the speedup.

The multitasking results obtained on the Cray X-MP/48 for 100 computational cells are shown in Table I. The simulated results are in good agreement with the actual results obtained in a dedicated environment; that is, with no other jobs running. As the number of processes increased, vector lenghts decreased. In order to see how big this effect was, the serial and multitasked codes were rerun with the vectorization turned off. These speedups were slightly higher than the vector speedups for four processors and about the same for two processors, but the times for running the same problem a number of times were inconsistent so the vector/no vector results are inconclusive. For four processors the times varied as much as 18 percent from the smallest to largest times for a series of runs. The reason is not yet known, but it maybe due to operating system interactions.

Table II shows the hypercube results for 100 and 1000 computational cells. Both actual results and predictions from Amdahl's law, with no overhead included, are shown. For 1000 computational cells, the predicted and actual efficiencies are in close agreement indicating that the overhead, which consists almost entirely of node to node communication is very small. For 100 cells, the overhead becomes a significant fraction of the total run time. For 100 cells and 32 nodes the predicted efficiency drops to about 38 percent if the ratio of communication to serial time is added to the denominator of Amdahl's law. Performance is obviously bad when the communication time becomes a large fraction of the total time (about 44 percent of the total time is in communication for 32 nodes and 100 cells).

Load balancing must be carefully considered in using the hypercube. The first method of load balancing considered involved dividing the total number of cells by the number of nodes and letting the last node do the remainder. For example, for 300 cells this method results in 15 nodes doing 18 cells and 1 node doing 30 cells. Figure 1 shows that this method of load balancing results in poor speedup when the number of cells is not close to a multiple of the number of nodes. A better method of load balancing would be to divide the cells so that no node has more than one cell more than any other node. Figures 2 and 3 show the efficiencies as a function of the number of nodes (processors) and as a function of the number of cells. for large problems the efficiency remains high even for 32 nodes.

Table III shows the Alliant results for one and six processors. As indicated previously, trivial changes were needed to multitask the code for this machine. A speedup of 3.87 on six processors was obtained for the 100 cell base case.

The RP3 is not yet running and therefore Table IV shows only simulated results. The simulator was run on an IBM 3081 under VS Fortran. The serial time is the actual 3081 time and not a simulated RP3 serial time. The speedups are thus relative values. The operating system can have problems when the number of virtual processors is greater than the number of physical processors (2). This can be seen in Table IV for eight processors where the efficiency exceeds the five processor value, which would not be expected.

A number of observations emerged from this work. The hypercube with only local memory required a substantial amount of additional coding for

message passing. It would be desirable to have at least global memory between pairs of nodes in order to pass boundary conditions from node to node. This feature will be included in the RP3. It would also be more convenient if I/O could be done from the hypercube nodes rather than just from the host. (I am not considering writing from the nodes to a log file since this does not work if a system program has not been started.)

The automatic parallelization features of the Alliant save large amounts of time and hopefully will be in other systems in the future.

Significant work is needed to make debugging easier, especially for large production codes, which we are currently multitasking. Local memory codes are easier to debug than global memory codes and are certainly easier to think about during the development phase.

In summary, a one-dimensional, time-dependent Lagrangian hydrodynamics code involving the solution of three coupled hyperbolic partial differential equations has been multitasked. The Cray actual results are in good agreement with simulator results. This allows predictions to be made for future Cray-like machines with more than four processors. The effect of decreasing vector lengths appears to be small, at least for the test case considered. When communication times are included Amdahl's law gives an indication of how many hypercube nodes can be efficiently used.

TABLE I

ACTUAL AND SIMULATED RESULTS FOR CRAY X-MP/48
100 COMPUTATIONAL CELLS: VECTOR CODE

Number of Processors	Actual Speedup	Results Efficiency (%)	Simulated Speedup	Results Efficiency
2	1.72	86	1.69	84.5
4	2.92	73	2.87	71.8
8			4.39	54.8
16			5.15	32.2

TABLE II

1-D HYDRO CODE CUBE EFFICIENCY

Number of Nodes	Efficiency (%)			
	Maximum Predicted		Actual	
	100 cells	1000 cells	100 cells	1000 cells
2	98.96	99.14	93.4	98.0
4	96.95	97.46	81.2	96.2
8	93.15	94.26	68.1	93.6
16	86.39	88.46	49.7	87.6
32	75.44	78.76	32.9	76.2

TABLE III

TIMING RESULTS FOR ALLIANT*

Number of Processors	Time (sec)	Time for One Processor Time	Percent Efficiency
1	72.4	1	100
6	18.7	3.87	64.5

* These results were obtained by Dr. Olaf Lubeck, Computer Research and Applications Group, Los Alamos National Laboratory.

TABLE IV

SIMULATED TIMING RESULTS FOR IBM RP3*

Code	Number of Processors	Time (sec)	Serial Time / Time	Percent Efficiency
Serial	1	44.4	1	100
Multitasked	1	54.0	.82	82
	2	33.0	1.34	67
	4	17.8	2.49	62
	5	16.9	2.62	52.5
	8	9.1	4.88	61.0
	16	6.5	6.83	43.0

*Dr. Frederica Darema - Rogers, IBM Thomas J. Watson Research Center converted the code for the RP3 simulator and obtained the above results.

REFERENCES

1. E. Williams and F. Bobrowicz, "Speedup Predictions For Large Scientific Parallel Programs on Cray X-MP-Like Architectures," Proceedings of the 1985 International Conference on Parallel Processing, St. Charles, Illinois, August 20-23, 1985.

2. Multitasking User Guide, Cray Computer Systems Technical Note SN-0222, Cray Research, Inc., Mendota Heights, Minnesota, (1984).

3. IPSC User's Guide, Intel Corporation, Order Number: 175455-003, (October, 1985).

4. Intel Fortran-286 User's Guide for XENIX 286 Systems, Intel Corporation 122196-001 (1985).

5. Alliant FX/Series Product Summary, Alliant Computer Systems Corporation, 42 Nagog Park, Acton, Mass. 01720, June, 1985.

6. G. F. Pfister et al., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," Proceedings of the 1985, International Conference on Parallel Processing, St. Charles, Illinois, August 20-23, 1985.

7. J. M. Stone, F. Darema-Rogers, V. A. Norton, and G. F. Pfister, "Introduction to VM/EPEX Fortran Preprocessor," IBM T. J. Watson Research Center, Yorktown Heights, New York, RC 11407 (#51329), September 30, 1985.

8. Robert D. Richtmyer and K. W. Morton, "Difference Methods for Initial-Value Problems," Interscience Publishers, New York (1967).

9. John K. Dukowicz, "A General, Non-Iterative Riemann Solver for Godunov's Method," J. Comp. Phy., 61 (1985), 119-137.

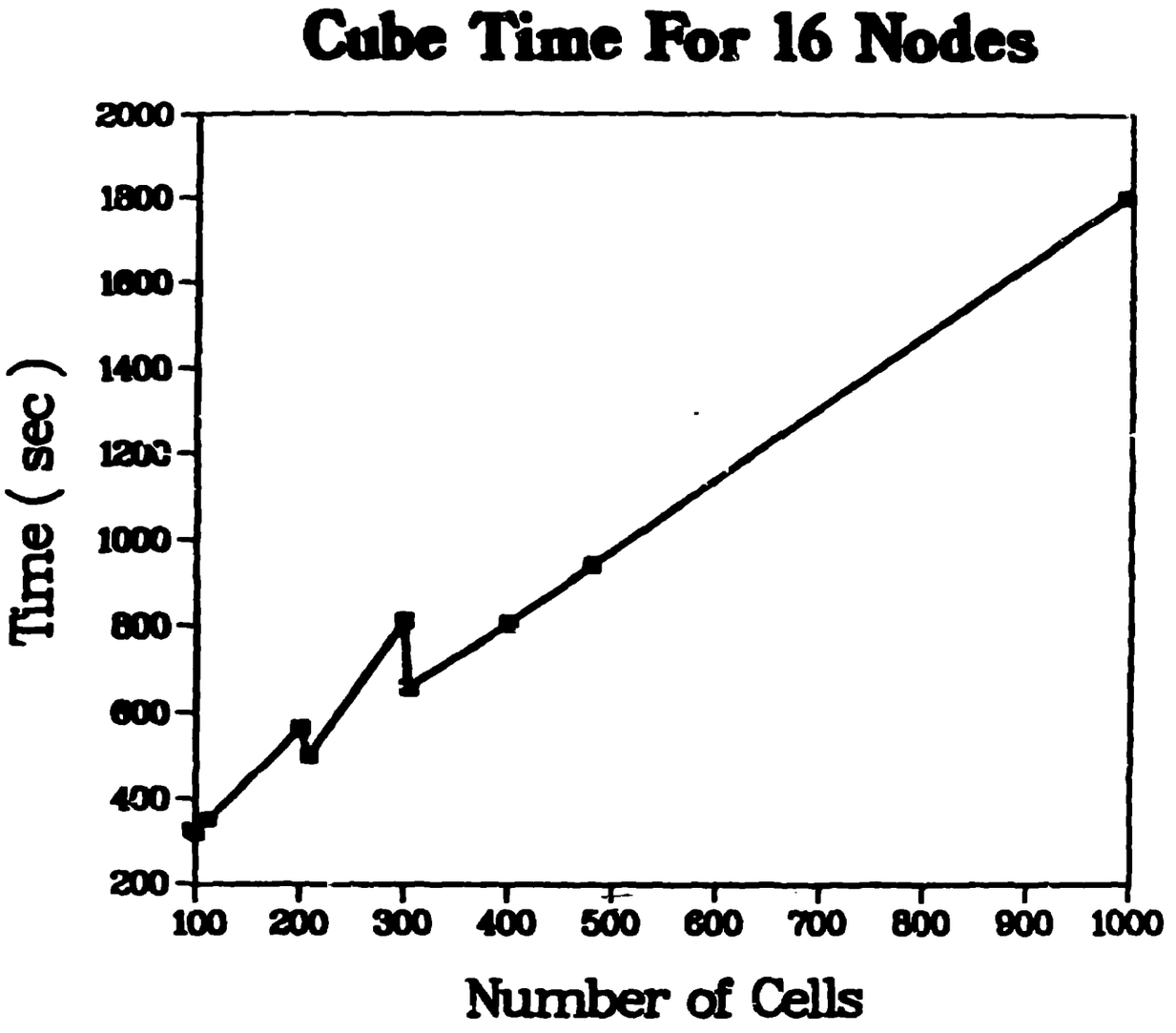

Fig. 1 Effect of Load Balancing on Hypercube Results

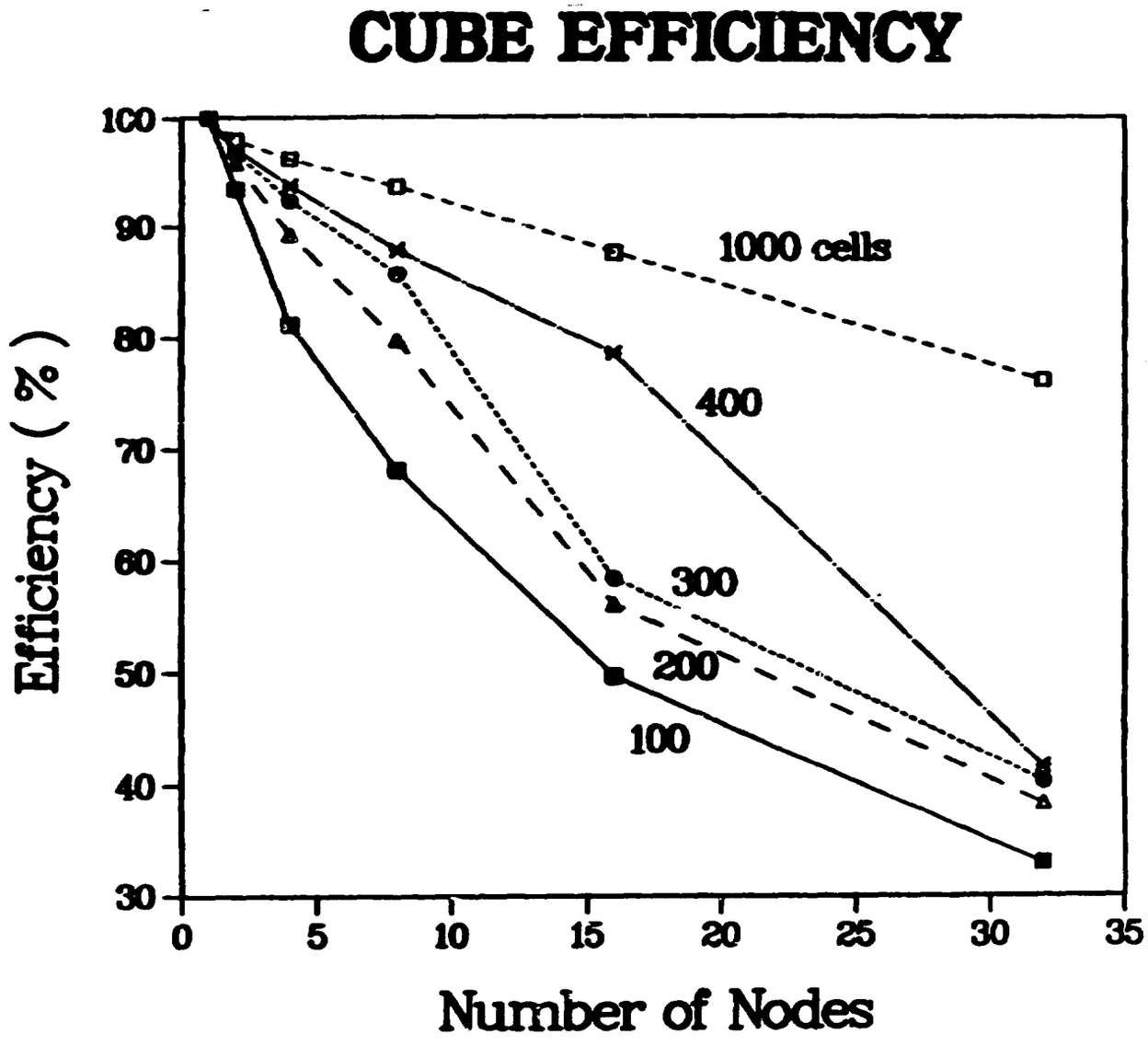

Fig. 2 Cube Efficiency vs number of Nodes

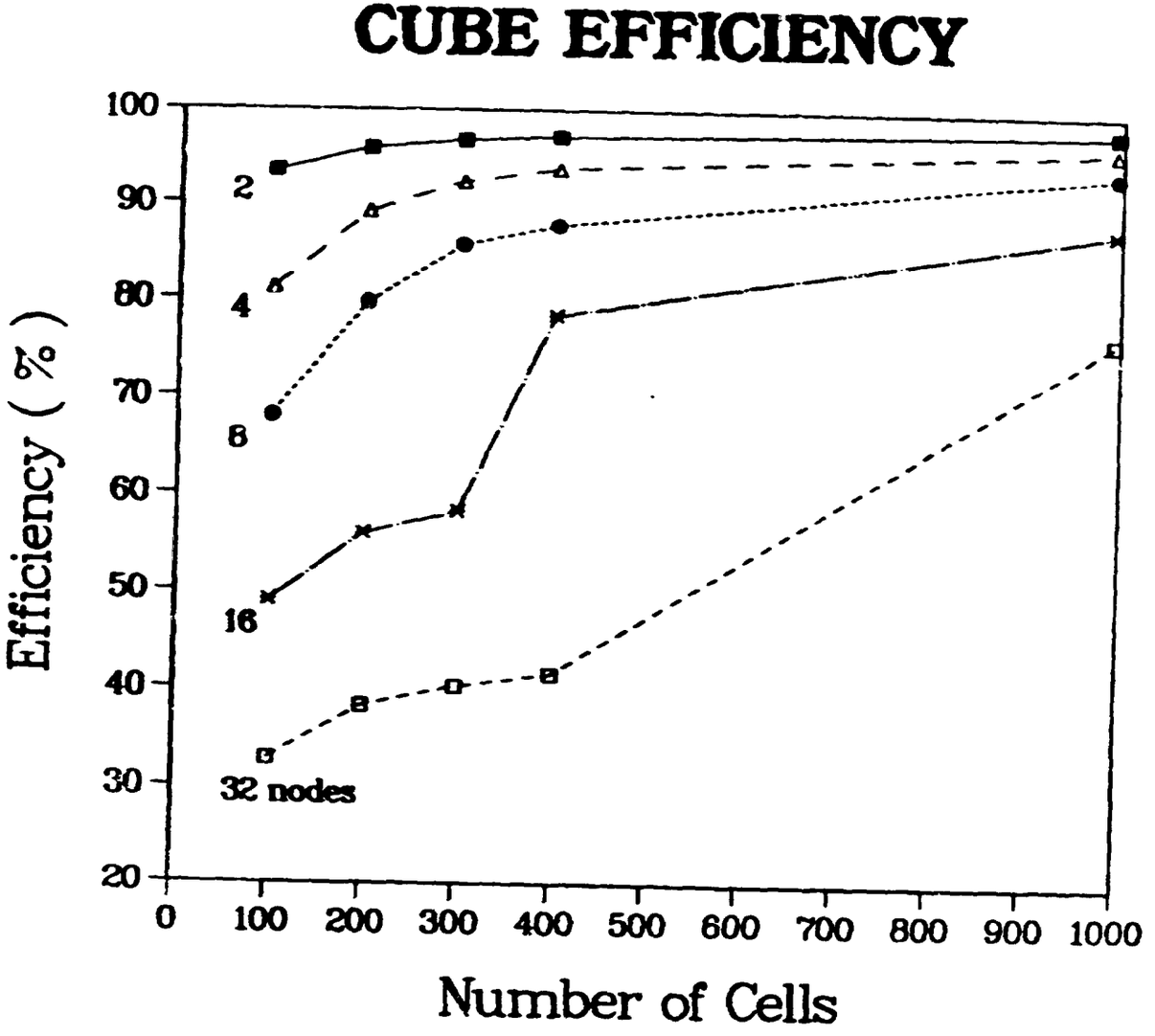

Fig. 3 Cube Efficiency vs Number of Computational Cells