Implicit Finite Element Applications: A Case for Matching the Number of Processors to the Dynamics of the Program Execution*

Meenakshi A.Kandaswamy † Valerie E. Taylor ‡ Rudolf Eigenmann §
Jose' A. B. Fortes§

Abstract

Generally, parallel scientific applications are executed on a fixed number of processors determined to be optimal by an efficiency analysis of the application's computational kernel. It is well-known, however, that the degree of parallelism found in different parts of an application varies. In this paper, we present the results of an in-depth study *quantifying* the advantages of matching the number of processors to the parallelism profile for a widely used application, finite element analysis. The study entails using *effectiveness* as the performance metric. The results indicate that the varying processor allocation is significantly more effective than fixed processor allocation by one to nine orders of magnitude for problems with 10^5 to 10^9 nodes. Further, the results indicate that it is very effective to match the number of processors to the parallelism profile even when only a small percentage of the application has a different degree of parallelism.

1 Introduction

Generally, parallel scientific applications are executed on a given number of processors, which remains fixed throughout execution. This number is often determined by analyzing the computational kernel of the application in terms of efficiency. In reality, however, the degree of parallelism of the application is a value which varies dynamically throughout the execution of the application. While it is well-known that the degree of parallelism varies, the advantages of matching the processor allocation to the parallelism profile has not been quantified for large problems and large number of processors. In this work, we quantify the advantages of allowing the number of processors to vary throughout the execution of a widely used scientific application. An example of such a machine would be a multiprocessor that allows multiprogramming, thereby enabling different users to simultaneously use different number of processors throughout the execution of their programs; the total number of processors in use at any given time remains fixed.

The application that is used in our study is the Fast Implicit Finite Element Analysis (FIFEA) code [2]. FIFEA is a parallel implementation that provides the framework for

*This research was supported in part by NSF grants ASC-9612133 and ASC-9612023

†CIS Dept., Syracuse University, Syracuse, NY 13244 e-mail: meena@ece.nwu.edu (This work was done at Northwestern University.)

‡ECE Dept., Northwestern University, 2145, Sheridan Road, Evanston, IL 60208 e-mail: taylor@ece.nwu.edu

§School of ECE, 336 EE Bldg. West Lafayette, IN 47907 e-mail:[eigenman, fortes]@ecn.purdue.edu

the analysis of the finite element method, which is used widely in various engineering and scientific problems such as biomedical engineering or structural analysis. The comparison between using a fixed number of processors and matching the number of processors to the parallelism profile is accomplished by estimating the execution time via an extrapolation from the timings obtained from the actual execution on the IBM SP. The estimation is such that the relative timings of the different steps of FIFEA is consistent with that observed on the IBM SP. The FIFEA code is analyzed for problem sizes ranging from 10^5 to 10^9 nodes. The computation to communication ratio is equal to 10^{-3}, which reflects the processor and network technologies currently found in parallel computers including the IBM SP. The number of processors varies from 1 to the problem size. For the varying number of processors case, the suitable number of processors for each step is selected if the efficiency of going to a higher number of processors is at least 40%.

The performance metric used to compare the fixed with the varying processor allocations is *effectiveness*, as defined in [10]. Good effectiveness, which is defined and explained further in Section 4, indicates good speedup with a small parallel cost (i.e., small overhead). The results of the analysis indicate that the varying processor allocation provides three to five orders of magnitude better effectiveness than the fixed processor allocation for the 10^5 node problem and one to nine orders of magnitude better effectiveness for the 10^9 node problem. Further, for the 10^5 node problem, the higher effectiveness resulted when 93.3% of the execution time was assigned to 500 processors and the remaining 6.67% was assigned to 1, 200 and 50,000 processors. Similar trends occurred with the other problem sizes.

2 FIFEA

FIFEA makes extensive use of the PETSc (Portable, Extensible, Toolkit for Scientific computation) library [1] for linear algebra and to manipulate sparse matrices and vectors. FIFEA and PETSc use the BlockSolve95 library[6] to store the sparse matrices and the MPI library [5] for communication between processors. The particular application of FIFEA used in this study was the analysis of an automotive disk brake system. The analysis entailed collecting performance data (i.e., timings of the various computation and communication steps) of the execution of the application on the IBM SP system available at Argonne National Laboratory; the timing data was collected using the Pablo software environment [9]. In addition, the code was hand analyzed to identify the main computational phases and the various computational steps in each phase.

Four main phases of FIFEA were identified: setup, initialization, static, and dynamic. In the setup and initialization phases, various data structures necessary for the finite element solution steps are set-up and initialized. These two phases include the following operations: creating lists of faces, calculating body inertias, assembling the diagonal mass matrix, initializing rigid body coordinates, computing rigid body acceleration, and creating vectors for the solution step.

Both the static and dynamic phases employ the preconditioned conjugate gradient solver, with the preconditioner being the Incomplete Cholesky factorization. The solver is efficiently implemented in parallel in the BlockSolve95 library, which uses a parallel heuristic [7] to color the graph. This coloring presents an ordering of the nodes to be computed in parallel in both the factorization and the forward and backward substitution steps. In the static phase, the solver step is executed for one instance in time. The preconditioned conjugate gradient solver consists of various operations including SAXPY, vector dot products, matrix vector multiplication, forward substitution, and back substitution. In

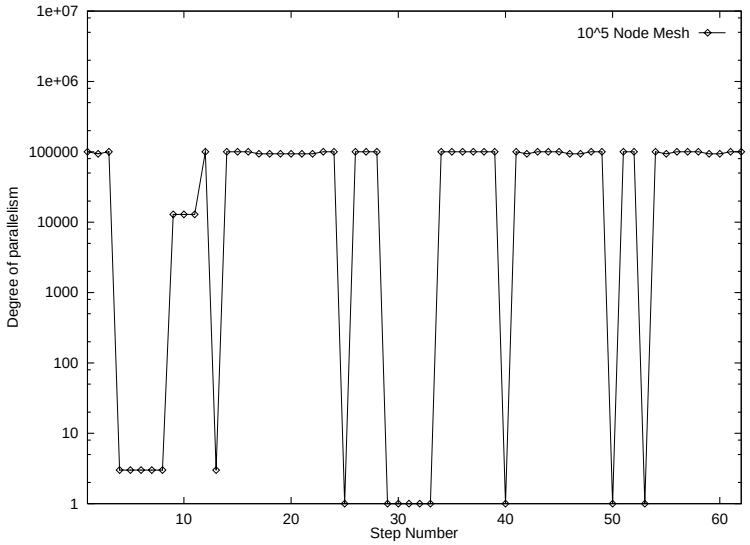

FIG. 1. *Parallelism profile for* 10^5 *nodes.*

the dynamic phase, the solver is executed in a main loop, which iterates over different time steps.

Figure 1 gives the parallelism profile of an instance of the FIFEA code for a problem size of 10^5 nodes. The degree of parallelism is given for the 63 steps identified in the FIFEA code. This data is based upon the hand analysis. Steps 1–19 comprise the set-up phase, steps 20–39 comprise the initialization phase, steps 40–49 comprise the static phase, and steps 50–63 comprise the dynamic phase. The FIFEA code executes the steps in sequence with steps 50–63 executed for different time intervals. The results indicate that the degree of parallelism varies throughout execution, with the maximum equal to the problem size. Similar results occurred with the other problem sizes.

3 Performance Metrics

Efficiency is used to identify the number of processors to be used for each step. Initially, a step is assigned the lowest number of processors and then the number of processors is increased only when the efficiency of using a larger number of processors is at least 40%. This scheme attempts to match the number of processors to the parallelism profile of the application for efficient execution.

The performance metric, effectiveness, is used to compare the fixed processors allocation with the varying processor allocation cases. The term F_P, effectiveness for P processors, is efficiency per time unit and can be stated as:

$$F_P = \frac{E_P}{T_P} = \frac{T_1}{((T_P * P) * T_P)} = \frac{E_P S_P}{T_1} \tag{1}$$

where,

- $P \rightarrow$ the number of processors on which the parallel algorithm is executed
- $T_1 \rightarrow$ the time taken by the sequential algorithm
- $T_P \rightarrow$ the time taken by the parallel algorithm when executed on P processors

Effectiveness is a combined measure of speedup and efficiency. It is often the case with parallel scientific applications that increases in speedup occur at the cost of decreases in

efficiency. The goal is to maximize effectiveness, which implies maximizing efficiency while decreasing parallel execution time; small parallel execution time leads to good speedup. To see this, effectiveness can also be expressed as:

$$F_P = \frac{S_P}{PT_P} \tag{2}$$

where $P * T_P$ is the 'cost' of the parallel implementation. Equation 2 implies that good effectiveness implies large speedup at a small parallel cost. Similarly, from Equation 1, we can say that maximizing effectiveness will result in both high efficiencies and low execution times. Hence, achieving good effectiveness results in achieving both good efficiency and speedup.

As effectiveness measures efficiency per time unit, good effectiveness ensures that the program maintains good utilization of processors throughout the execution of the program.

4 Analysis of Results

The FIFEA application was analyzed for problem sizes ranging from 10^5 to 10^9 nodes. The number of processors varied from 1 to the problem size, which is the maximum degree of parallelism in FIFEA as indicated in Section 2. The analysis entails estimating the execution time for a given problem size executing on a specified number of processors and using the estimation to calculate efficiency and effectiveness. The execution time was estimated via an extrapolation from the timings obtained from the actual execution on the IBM SP. The estimation was such that the relative timings of the different steps of FIFEA was consistent with that observed on the IBM SP. For our estimation, we fixed the computation to communication ratio at 10^{-3}, which reflects the processor and network technologies currently found in parallel computers, including the IBM SP.

4.1 Effectiveness

The values of estimated execution time, efficiency, and effectiveness for various fixed number of processors and varying number of processors are given in Tables 1, 2 and 3 for problem sizes of 10^5, 10^7 and 10^9 nodes, respectively. As the varying processor case uses different number of processors for different steps, the efficiency and effectiveness is a weighted sum where the values for each step are weighted by each step's respective contribution to the execution time. The value of effectiveness for the varying processor case is three to five orders of magnitude greater than the fixed processor cases for the 10^5 node problem, two to five orders of magnitude greater for the 10^7 node problem, and one to nine orders of magnitude for the 10^9 node problem. Hence, the varying processor case has the most effective parallel implementation of all the cases.

In some cases, the efficiency of the fixed processor allocation is greater than that of the varying processor allocation. In these cases, however, the estimated execution time is greater than that for the varying processor case because some parts of the program have higher degree of parallelism than the fixed number of processors. The result is a smaller effectiveness for these cases. For example if we consider the problem size of 10^9 nodes, the fixed processor case of 500 processors has a greater efficiency and estimated execution time than the varying processor case. The result, for this case, is that the effectiveness is two orders of magnitude smaller than the varying processors case.

Further, in some cases the estimated execution time is smaller for the fixed processor allocation; this occurs at the cost of small efficiency. Again, the result is a smaller

effectiveness. If we take the example of the 10^7 node problem, the estimated execution time of all the fixed processors cases, with the exception of the single and 500 processor case, is smaller than the varying processor case. Recall that the varying processor allocation selects a number of processors for each step based upon efficiency not execution time. Yet, for this problem size, the effectiveness for the fixed processors cases is two to five orders of magnitude smaller than that for the varying processor case.

TABLE 1
Analysis Results for Problem size of 10^5 *Nodes*

Num Procs	Estimated Execution Time	Efficiency	Effectiveness
1	1.73e+04	1.00e+00	5.77e-05
500	2.38e+02	1.46e-01	6.13e-04
1000	2.21e+02	7.82e-02	3.54e-04
4000	2.10e+02	2.06e-02	9.81e-05
6000	2.09e+02	1.38e-02	6.60e-05
8000	2.09e+02	1.04e-02	4.97e-05
10,000	2.09e+02	8.30e-03	3.98e-05
50,000	2.09e+02	1.66e-03	7.92e-06
100,000	2.09e+02	8.25e-04	3.93e-06
varying procs	2.29e+02	1.79e-01	1.41e-01

TABLE 2
Analysis Results for Problem size of 10^7 *Nodes*

Num Procs	Estimated Execution Time	Efficiency	Effectiveness
1	1.75e+06	1.00e+00	5.72e-07
500	4.50e+03	7.78e-01	1.73e-04
1000	2.75e+03	6.36e-01	2.31e-04
4000	1.44e+03	3.04e-01	2.11e-04
6000	1.29e+03	2.25e-01	1.74e-04
8000	1.22e+03	1.79e-01	1.46e-04
10,000	1.18e+03	1.48e-01	1.26e-04
50,000	1.04e+03	3.36e-02	3.23e-05
10,0000	1.02e+03	1.71e-02	1.67e-05
1,000,000	1.01e+03	1.73e-03	1.71e-06
10,000,000	1.01e+03	1.73e-04	1.71e-07
varying procs	3.54e+03	7.22e-01	1.29e-02

4.2 Estimated Execution Time Distribution

Tables 4 and 5 present the distribution of the estimated execution time across different groupings of processors for the varying processor allocation case. The results indicate that having only a small percentage of the execution time distributed across different groupings

TABLE 3
Analysis Results for Problem size of 10^9 *Nodes*

Num Procs	Estimated Execution Time	Efficiency	Effectiveness
1	1.75e+08	1	5.70e-09
500	3.69e+05	9.51e-01	2.58e-06
1000	1.93e+05	9.06e-01	4.68e-06
4000	6.20e+04	7.07e-01	1.14e-05
6000	4.74e+04	6.16e-01	1.30e-05
8000	4.01e+04	5.47e-01	1.36e-05
10,000	3.57e+04	4.91e-01	1.38e-05
50,000	2.17e+04	1.62e-01	7.46e-06
100,000	1.99e+04	8.80e-02	4.42e-06
1,000,000	1.83e+04	9.55e-03	5.21e-07
10,000,000	1.82e+04	9.63e-04	5.30e-08
100,000,000	1.82e+04	9.64e-05	5.30e-09
1,000,000,000	1.82e+04	9.64e-06	5.30e-10
varying procs	5.13e+04	6.46e-01	9.14e-04

of processors is very effective. For example, with the 10^5 node problem 93.33% of the execution occurs with the 500 processors and the remaining 6.67% is distributed among 1, 200 and 50,000 processors. In Table 1, for the 500 fixed processor case, the effectiveness is 6.13e-04; for the varying processor case the effectiveness is 1.41e-01. The varying processor case is three orders of magnitude more effective, yet only 6.67% of the execution makes use of other processor groupings. Similar results occur with the other problem sizes. Hence, it is effective to match the number of processors to the parallelism profile even when only a small percentage of the application has a different degree of parallelism.

TABLE 4
Processor distribution for the varying processor case : 10^5 *and* 10^6 *nodes problem size*

10^5		10^6	
# procs	% time	# procs	% time
1	6.55	1	2.49
500	93.33	500	97.24
2000	0.10	1000	0.16
50,000	0.02	9000	0.09
		500,000	0.01

5 Related Work

Other work in this area includes the development of a self-tuning runtime system and the design of a parallel machine with a hierarchy of processors. Nguyen et. al [8] have developed a runtime system that dynamically adjusts the number of processors used by the application based on dynamic measurements of performance gathered during execution. The runtime system has been developed in the context of shared memory multiprocessors. The metric used to determine processor allocation is speedup. Their self-tuning schemes are based on

TABLE 5
Processor distribution for the varying processor case : 10^7, 10^8 and 10^9 nodes problem size

10^7		10^8		10^9	
# procs	% time	# procs	% time	# procs	% time
1	0.42	1	0.08	1	0.029
500	98.59	500	21.78	500	35.17
1000	0.93	700	59.16	2000	50.06
10,000	0.02	1000	7.9	10,000	14.65
100,000	0.01	2000	10.86	100,000	0.08
5,000,000	0.002	10,000	0.18	1,000,000	0.01
		100,000	0.005	500,000,000	0.0001
		500,000	0.006		
		50,000,000	0.0004		

the golden sections method used in non-linear programming. The class of applications to which they can apply this technique have to be iterative in nature. Initial work on the KSR-2 running SPLASH and Perfect Club benchmarks demonstrate better performance than fixed processor allocation. The maximum number of processors used in this work was 50. Our work uses estimated execution time, based on an extrapolation, which allows us to consider large problems executed on large number of processors. Further, we are able to go in-depth and consider the execution time distribution.

Z. Ben Miled et. al. [3, 4] have proposed a a cost-effective multiprocessor architecture that takes into consideration the importance of hardware and software costs as well as delivered performance in the context of real applications. The proposed architecture, called HPAM, is organized as a hierarchy of processors-and-memory (PAM) subsystems. Each PAM contains one or more processors and one or more memory modules. An HPAM consists of several levels of processors-and-memory (PAM) systems. Relative to the first (top) level of the hierarchy, each additional level has an increasing number of slower processors. Each PAM system can be implemented with different processors and interconnections. Applications on the HPAM can be mapped to the different levels based on the resource requirements of the different phases of the application. Also HPAM is a multi-programmable architecture allowing different programs to simultaneously run using different number of processors.

6 Conclusion

In this paper we quantified the advantages of matching the processor allocation to the parallelism profile for an implicit finite element application. Large problems were analyzed and the maximum number of processors was allowed to equal the problem size. Effectiveness was chosen as a metric that can capture speedups and efficiencies of a parallel implementation. The results of our study show that varying the number of processors is the most effective for the implicit finite element application. In particular, the varying processor allocation scheme was three to five orders of magnitude more effective than the fixed processor allocation for the 10^5 node problem, two to five orders of magnitude for the 10^7 node problem, and one to nine orders of magnitude for the 10^9 node problem. Further, for the 10^5 node problem, the higher effectiveness resulted when 93.3% of the execution time was assigned to 500 processors and the remaining 6.67% was assigned to 1, 200 and 50,000 processors. Similar trends occurred with the other problem sizes. Hence, it is very effective to match the number of processors to the parallelism profile even when only a small

percentage of the execution has a different degree of parallelism. The study indicates that the number of processors used for execution should be matched to the parallelism profile of the application.

7 Acknowledgments

The authors would like to acknowledge Dr. Thomas Canfield, of Argonne National Laboratory, for the use of his code FIFEA and Shai Eisen, of Northwestern University, for assisting with collecting the performance data from the IBM SP. The authors also acknowledge Argonne National Laboratory for the use of their IBM SP machine.

References

[1] S.Balay, W.Gropp, L.Curfman McInnes, and B.Smith, "PETSc 2.0 Users Manual," *Technical Report ANL-95/11. Argonne National Laboratory.*, 1995.

[2] T. Canfield, T. Disz, M. Papka, R. Stevens, M. Huang, V. Taylor, and J. Chen, "Toward real-time interactive virtual prototyping of mechanical systems: Experiences coupling virtual reality with finite element analysis," *High Performance Computing Conference,* 1996.

[3] M.Kandaswamy, Z.Ben Miled, B.Armstrong, S.Kim, V.Taylor, R.Eigenmann and J.Fortes, "Progress towards the design of a hierarchical processors-and-memory architecture for high performance computing," *The Petaflop frontier workshop - Sixth symposium on the frontiers of massively parallel computation,* Oct 1996.

[4] Z. Ben Miled, J.A.B.Fortes, R.Eigenmann and V.Taylor, "Hierarchical processors-and-memory architecture for high performance computing," *Sixth symposium on the frontiers of massively parallel computation,* Oct 1996.

[5] W.Gropp, E.Lusk, and A.Skjellum, "Using MPI Portable Parallel Programming with the Message-Passing Interface," *Massachussets: MIT Press, Cambridge.*

[6] M.T.Jones and P.E.Plassman, "BlockSolve95 Users manual: Scalable library software for the parallel solution of sparse linear systems," *ANL Report 95/48.* Argonne National Laboratory.

[7] M.T.Jones and P.E.Plassman, "A Parallel graph coloring heuristic," *SIAM J. Scientific and statistical computing 14 654-669,* 1993.

[8] T.D.Nguyen, R.Vaswani and J.Zahorjan, "Maximizing speedup through self-tuning of processor allocation," *Technical Report 95-09-02, Department of computer science and engineering Univ. of . Washington, Seattle.*

[9] D. A. Reed, K. A. Shields, W. H. Scullin, L. F. Tavera, and C. L. Elford, "Virtual reality and parallel systems performance analysis," *IEEE Computer*, November 1995.

[10] U. Schendel, "Introduction to numerical methods for parallel computers," *Ellis Horwood Publishers, Chichester,* 1984.