Experimental Evaluation and Workload Characterization for High-Performance Computer Architectures

Tarek A. El-Ghazawi, P.I.
Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC 20052

Final Report
1/1/94-5/31/95

Submitted to USRA/CESDIS

This research is conducted in the context of the Joint NSF/NASA Initiative on Evaluation (JNNIE). JNNIE is an inter-agency research program that goes beyond typical benchmarking to provide and in-depth evaluations and understanding of the factors that limit the scalalabiltiy of high-performance computing systems. Many NSF and NASA centers have participated in the effort. Our research effort was an integral part of implementing JNNIE in the NASA ESS grand challenge applications context. Our research work under this program was composed of three distinct, but related activities. They include the evaluation of NASA ESS high-performance computing testbeds using the wavelet decomposition application; evaluation of NASA ESS testbeds using astrophysical simulation applications; and developing an experimental model for workload characterization for understanding workload requirements.

In this report, we provide a summary of findings that covers all three parts, a list of the publications that resulted from this effort, and three appendices with the details of each of the studies using a key publication developed under the respective work.

SUMMARY OF FINDINGS

Wavelet Decomposition On High-Performance Computers

In this study, we have mapped the multi-resolution Wavelet algorithm, developed by Mallat [Mal89], onto the high-performance parallel computers and applied it to remotely sensed data from NASA's Landsat-Thematic Mapper images. Target platforms were the ESS MasPar MP-1 and MP-2, and the ESS Intel Paragon. The MasPar has provided two orders of magnitude improvement over a workstation and exhibited good scalability. The Intel Paragon produced one order of magnitude improvement and required knowledge about the network operation and special effort to scale beyond 4 processors.

Astrophysical Simulations on High-Performance Computers

In this study some of the sources of overhead were identified and measured for Nbody and the Particle In Cell (PIC) ESS applications. Among the observed sources of overhead are the programming model, programming style, and the communications patterns. With the sophistication of multicomputers and in the light of the lack of comparably powerful compiler technology, parallel machines are much less forgiving than uniprocessor environments. Subtle changes in programs can increase or decrease overhead significantly. Some types of overhead can be reduced by following better programming practices and some can be reduced by converting them to less costly overhead activities. The dominant type of overhead is communications and it often represents a real challenge to scalability. While it is not considered a good programming practice, duplication redundancy can effectively help reduce the effect of communications. Efficiency in most of the cases, specially when data sets were large enough, was greater than 50% which indicates that progress in high-performance computing is consistent with the needs of scientific parallel simulations.

Parallel Workload Characterization

This study introduced a parallelism-based methodology for an easy to understand representation of workloads. The method is architecture-invariant and can be used effectively for the comparison of workloads and assessing resource requirements. A method for comparing workloads based on the notion of centroid of parallel instruction was introduced. This method uses the normalized Euclidean distance to provide an efficient means of comparing the workloads. The notion of centroid coupled with distance (similarity) among pairs of workloads provide the basis for quantifiable analysis of workloads to make informed decisions on the composition of parallel benchmark suites. Analysis of existing benchmarks is also provided for by this model in which the centroid sheds light on the hardware resource requirements and how the benchmark is exercising the target machines, and the distance among workload pairs allows identifying possible correlation.

LIST OF PUBLICATIONS FROM THIS EFFORT

Papers

[1] T. El-Ghazawi and Jacqueline Le Moigne. "Mutiresolution Wavelet Decomposition on the MasPar Massively Parallel System". Journal of Computers and Their Applications, Vol. 1, NO. 1, August 1994.

[2] Chan, Chui, LeMoigne, Lee, Liu, and El-Ghazawi. "The Performance Impact of Data Placement for Wavelet Decomposition of Two Dimensional Image Data on SIMD Machines". Proceedings of Frontiers'95, McLean, IEEE CS Press, February 1995.

[3] A. Ozkaya and T. El-Ghazawi. "An Electrostatic Particle_In_Cell (PIC) Simulations on the Intel Paragon". Proceedings of the Parallel and Distributed Computing and Systems, Orlando, September 1995.

[4] T. Sterling, S. Zalasak, and T. El-Ghazawi. "An Innovative Approach to Benchmarking Scalable Parallel Computers for the Earth and Space Sciences Problem Domain". Scalable High-Performance Computing Conference'94 (poster session), IEEE Computer Society, Knoxville, TN, May 1994.

CESDIS Technical Reports

[5] T. El-Ghazawi and Jacqueline Le Moigne. "Mutiresolution Wavelet Decomposition on the MasPar Massively Parallel System". TR-94-122.

[6] Chan, Chui, LeMoigne, Lee, Liu, and El-Ghazawi. "The Performance Impact of Data Placement for Wavelet Decomposition of Two Dimensional Image Data on SIMD Machines". TR-94-125.

APPENDIX A:

WAVELET DECOMPOSITION ON HIGH-PERFORMANCE COMPUTERS

In Collaboration with Dr. Jacqueline Le Moigne (CESDIS)

Wavelet Decomposition on High-Performance Computing Systems

Tarek A. El-Ghazawi

Department of Electrical Engineering and Computer Science
The George Washington University
Washington, D.C. 20052
tarek@seas.gwu.edu
(202)994-5507

Jacqueline Le Moigne

Center of Excellence in Space Data and Information Sciences
NASA Goddard Space Flight Center
Code 930.5
Greenbelt, MD 20771

Abstract

Wavelet transforms are distinguished with many favorable characteristics, over other popular signal processing techniques such as the Fourier transform. Due to the recent discovery of a fast sequential algorithm for Wavelet, by Mallat, the past few years have seen a remarkable increase in applying Wavelets to real-life problems, in which speed is critical. In this paper we present and compare efficient Wavelet decomposition algorithms on different parallel architectures. We report and analyze experimental measurements, using NASA remotely sensed images. The results show that the algorithms achieve significant performance gains on current high-performance parallel systems and meet scientific applications and multimedia requirements.

Index Terms: Parallel Processing, Experimental Performance, Image Processing, and Wavelets

1. Introduction

Traditionally, Fourier transforms have been utilized for signal analysis and reconstruction. However, Fourier representations do not include any local information about the original signals. Therefore, windowed Fourier transforms, particularly Gabor transforms [Chu92], have been introduced. With a windowed Fourier transform, a signal is analyzed after filtering by a fixed window function. Such window functions produce the localization effect that traditional Fourier transforms lack. However, since the envelope of the signal is the same for all frequencies, a windowed Fourier transform uniformly samples the time-frequency plane. Depending on the applications, for example speech analysis or image feature extraction, it can be of interest to have a more flexible division of the time-frequency plane to provide more "time-details" for high frequencies. Wavelet transforms provide this type of sampling by filtering the signal with translations and dilations of a basic function, called "mother Wavelet".

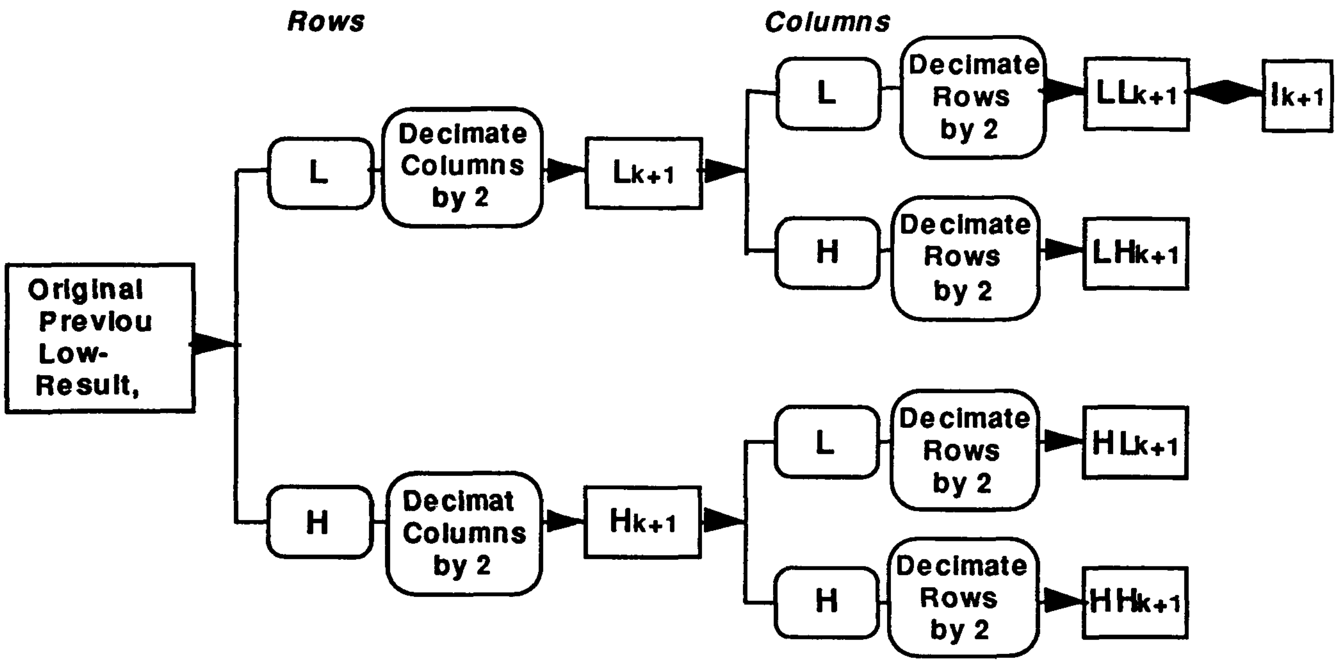
In the image processing domain, Wavelet transforms have been proven to be very useful for such tasks as image compression and reconstruction, feature extraction, and image registration [Chu92, Dja92, Dau92, Mal89, Str89, Lem94]. Furthermore, the multi-resolution scheme developed by Mallat [Mal89, Cod92, Num92] provides a very fast algorithm which increases the importance of wavelets for on-line processing of imagery data. The speed of such processing is especially important for managing remotely sensed data whose already massive amount will grow even bigger with such programs as NASA's Earth Observing System (EOS).

In this study, we are investigating the parallel implementation and performance of the Mallat algorithm on parallel architectures. Coarse-grain algorithms for both the Intel Paragon are developed. Measurements are collected, analyzed and compared with the fine-grain MasPar experimental results [Chan95, El-Ghaz94]. Test image data from NASA's Landsat-Thematic Mapper (TM), were used. The results will show that the proposed algorithms can achieve orders of magnitude performance improvement on contemporary high-performance computing systems, when compared typical desktop workstations. Such performance can satisfy real-time image processing needed for large scientific databases, such as NASA's EOS Data Information System (EOSDIS), and multimedia applications. This paper is organized as follows. Section 2 provides an overview of the discrete Wavelet transform and the Mallat algorithm. Section 3 provides an overview to the Jet Propulsion Lab (JPL) Intel Paragon architecture, that was used in this study. Section 4 discusses the algorithms and implementation issues on the different high-performance computing architectures. Timing results and performance analysis are given in section 5. Conclusions are given in section 6.

2. Multi-Resolution Wavelet Decomposition

As described in section 1, a Wavelet transform is defined by the translations and the dilations of a basic function called "mother Wavelet." Depending on the application, continuous or discrete transforms may be utilized. Also, special conditions can be imposed on the mother Wavelet which leading to orthonormal bases of wavelets, which is particularly useful for data reconstruction [Dau92]. In this paper, we will only consider Wavelet transforms for the processing and analysis of 2-D image data. Thus, discussion will be limited to discrete wavelets, particularly those with orthonormal bases.

According to Mallat [Mal89], an orthonormal basis of wavelets can be defined by a scaling function and its corresponding conjugate filter L. In this case, the Wavelet decomposition of an image is similar to a quadrature mirror filter decomposition with the low-pass filter L and its mirror high-pass filter H. The decomposition of a 2-D image also assumes that the multi-resolution representation of the image space is "separable". This means that the two axes x and y can be treated independently in the decomposition as well as in the reconstruction. This decomposition is summarized in figure 1.



Multi-Resolution Wavelet Decomposition

<u>**Figure 1**</u>

The input image is first convolved along the rows by the two filters L and H, and the horizontal dimension of these two intermediate results is decimated by 2. Each of the two "column-decimated" images, L_{k+1} and H_{k+1}, is then convolved along the columns by the two filters L and H and decimated along the rows by two. This decomposition results into four images, LL_{k+1}, LH_{k+1}, HL_{k+1} and HH_{k+1}. Each of these images, such as the low/low image, LL_{k+1}, is taken as the new input to perform the next level of decomposition and so on.

The Multi-Resolution Wavelet decomposition algorithm can be described by the following sequence of steps:

(0) Start from the image I_0, level 0 of the multi-resolution sequence (k=0).

(1) High-Pass and low-pass filterings of image rows at level k.

(2) Decimate by 2 the number of columns: results in and L_{k+1} and H_{k+1}.

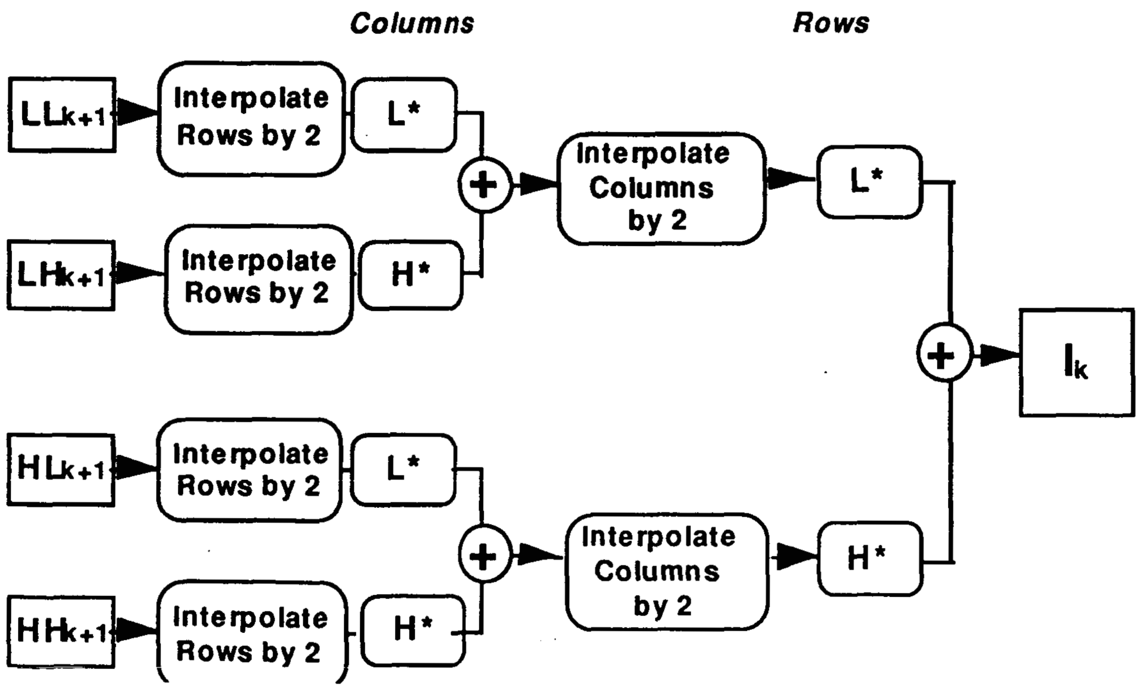
(3) High-Pass and low-pass filterings of image columns at level k.

(4) Decimate by 2 the number of columns: results in LL_{k+1}, LH_{k+1}, HL_{k+1}, and HH_{k+1}.

The low/low result, LL_{k+1} can be renamed I_{k+1}, since it corresponds to the compression of the original image at level k+1.

(5) Set k to the next level of decomposition, k+1, and continue the iterative process from (1) to (4) until the desired level of decomposition is achieved.

Wavelet reconstruction is obtained by a similar reverse process, which is graphically described in figure 2.



Multi-Resolution Wavelet Reconstruction

Figure 2

3. Overview of the Parallel Systems

Experimental measurements for this work were obtained using the NASA Earth and Space Science (ESS) high-performance computing testbeds. In particular, the GSFC MasPar MP-2 and the Jet Propulsions Lab (JPL) Intel Paragon were used. A brief description of these systems is given below.

3.1 The MasPar

MasPar Computer Corporation currently produces two families of massively parallel-processor computers, namely the MP-1 and the MP-2. Both systems are essentially similar, except that the second generation (MP-2) uses 32-bit RISC processors instead of the 4-bit processors used in MP-1. The MasPar MP-1 (MP-2) is a fine-grained, massively parallel computer with Single Instruction Multiple Data (SIMD) architecture. The MasPar has up to 16,384 parallel processing elements (PEs) arranged in a 128x128 array, operating under the control of a central array control unit (ACU). The processors are interconnected via the X-net into a 2-D mesh with diagonal and toroidal connections. In addition, a multistage interconnection network called the global router (GR) uses circuit switching for fast point-to-point and permutation transactions between distant processors. A data broadcasting facility is also provided between the ACU and the PEs. Every 4x4 grid of PEs constitutes a cluster which shares a serial connection into the global router. For more information on the MasPar, the reader can consult more specialized MasPar references [Bla90], [Mas92], [Nic90].

3.2 The Intel Paragon

The Paragon has a total of 64 nodes organized into a 16x4 mesh, of which 54 are compute nodes and 8 are service nodes. Each node, an Intel GP node, is essentially a separate computer with one compute and one communication i860 processors. Each of the 56 compute nodes has 32 MBytes of memory. The service nodes include: 4 I/O nodes with 32 MBytes memory and a 4.8 Gbyte RAID each, 1 HIPPI node with 32 MBytes memory, 1 User Service node with 32 MBytes memory, and 1 boot node with 32 MBytes memory and a 4.8 Gbyte RAID. The peak performance (using 56 nodes) is 5.6 GFlops in single precision with an aggregate memory space of 1.8 GBytes and aggregate online disk capacity in excess of 20 GBytes. The programs can be developed in C or FORTRAN which are supported by NX library routines for communication and synchronization purposes.

4. Parallel Implementations

In order to allow accurate measurements of communications, the message passing programming model was used in the Paragon. All applications were developed in C and augmented with PVM communication calls. The applications used the "single program, multiple data" (SPMD) programming model. In this model, the same program runs on each node in the application, but each node works on a part of the data. However, because each node is an independent computer, one can also use other programming models. One example is the "manager-worker" model, in which a "manager" program starts up several "worker" programs on other nodes, then gathers and interprets their results.

According to the previous descriptions, the Wavelet algorithm can be defined as a combination of successive *filterings* and *decimations*. Our parallel implementation will concentrate on these two operations, focusing on minimizing the communication costs by reducing the number of communication transactions and the distance between the communicating processors.

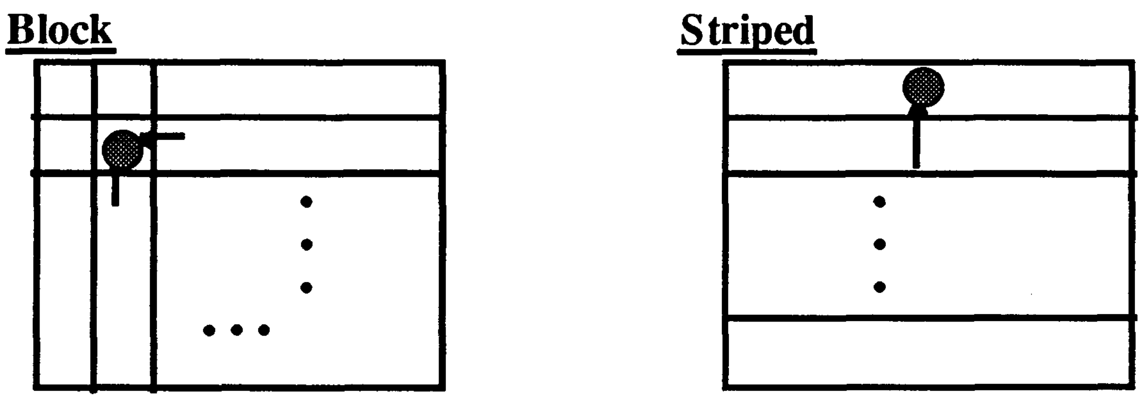
4.1 The Fine-Grain SIMD Implementations

On the MasPar MP-2, two algorithms were used, referred to as systolic and systolic with dilution, See [El-Ghaz94] and [Chan95] for details. Both of them store the filter in the control unit and broadcast the filter elements from last to first. After each broadcast, the algorithm requires one multiply and accumulate, followed by shifting the partial result to the left. The algorithm repeats this step for as many times as the size of the filter with partial results being accumulated and built up in a systolic fashion. By the last step, each (logical) processor ends up with one pixel result. The difference between the two algorithms is in the way decimation is handled. In the systolic algorithm, decimation is accomplished using the global router. In the dilution algorithm, the filter is diluted or stretched to be aligned with the relevant pixels, thus avoiding the use of the MasPar global router.

When the image data is larger than this basic size, a "virtualization" of the PE array has to be defined. Two virtualization methods were considered, “cut and stack” and hierarchical. The hierarchical gave the best results since it improves data locality for the underlying computations [Chan95]. In the "cut and stack" virtualization scheme, the image is cut into squares corresponding to the size of the basic parallel array. For example, if the size of the image is 512x512, we will need to stack 16 layers of image data in the 128x128 parallel array. The hierarchical virtualization divides up the image into subimages and allocates each subimage to a different physical processor. The MasPar systolic algorithm was shown to be processor optimal[El-Ghaz94].

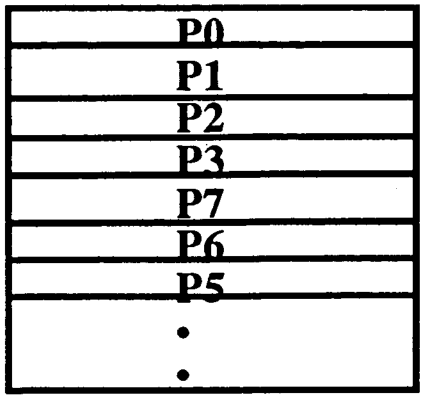
4.2 The Coarse-Grain MIMD Implementations

Reducing the number of transactions was done by distributing stripes of the image rather than blocks limiting exchange of information to one neighbor instead of two, which would have been needed should image data be distributed by blocks, see figure 3. Secondly, as seen in figure 4, those slices are distributed in a snake-like fashion in order to limit communications to immediate neighbors only. Those communications transactions are needed at the end of each decomposition level in order to build a guard zone around the processor local data from the decomposition results in its neighbors before the next decomposition level starts. Using a striped data decomposition, such zone is only needed for column filtering. In block data decomposition, guard zones need to

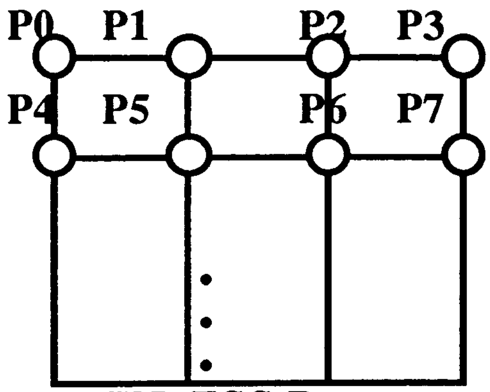


Reducing Communication Transactions Via Striping

Figure 3



Allocating image sub-domains to Processors



JPL ESS Paragon

Reducing the Paragon Communications Distances Via a Snake-Like Domain Decomposition

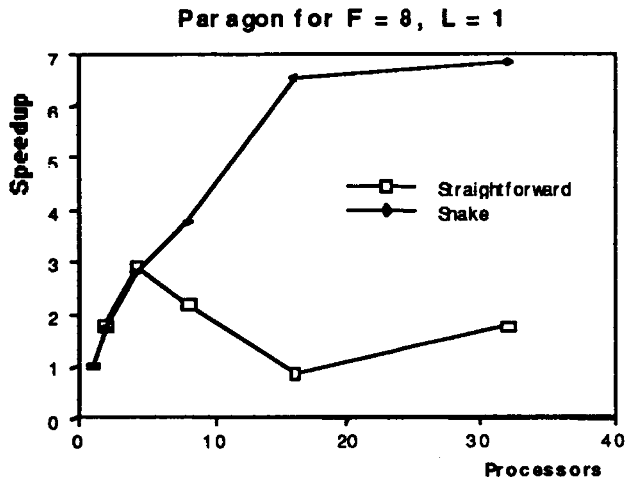
Figure 4

be established for both the row and column filtering. The depth of the zone is in the order of the filter length. Guard zone data is brought from the east neighbor in for row filtering, and from the south neighbor for the column filter.

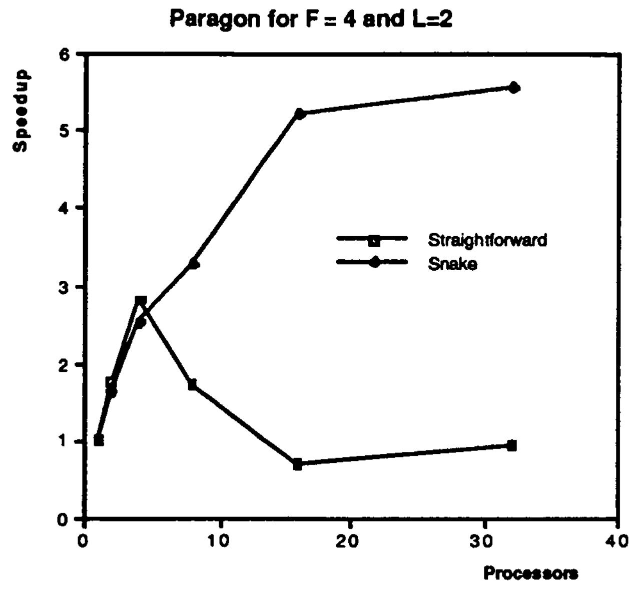
5. Experimental Results

Wavelet decomposition of a 512x512 Landsat-Thematic Mapper image of the Pacific Northwest area was used for our experiments. The experimental results for this image are given when filters of sizes 8, 4, and 2 are used along with 1, 2, and 4 levels of decompositions, respectively. It should be noted that as the number of decomposition levels increases, more communication is required. Increasing the filter size, however, increases the computational dominance in this problem.

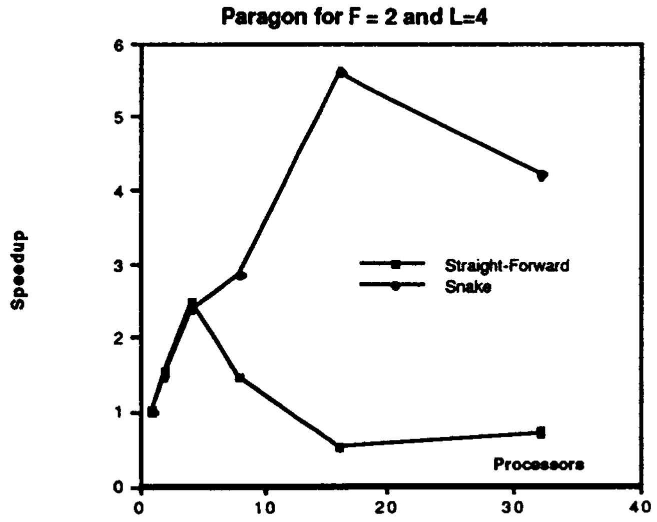
5.1 Intel Paragon Results



Paragon Performance for Filter Size 8 and 1 level of Decomposition
<u>**Figure 5**</u>



Paragon Performance for Filter Size 4 and 2 levels of Decomposition
Figure 6



Paragon Performance for Filter Size 2 and 4 levels of Decomposition

Figure 7

The Paragon scaling results are shown in figures 5 through 7. Scalability till 4 processors were obtained using the straight forward data distribution, where no arrangement was made to limit communication to nearest neighbors. The reason for the 4 can be seen from figure 4. Beyond 4 processors, processors at the right edge of the network attempt to communicate with those in the leftmost column of the following row. Due to dimension routing, where messages in this case would travel along the horizontal dimension first before moving along the vertical, conflicts would be created. For the small amount of computations in the Wavelet operations, this creates an excessive communications overhead that prevents scalability.

The snake-like data distribution on the other hand does not create these conflicts and limit communication to a distance of one, thus creating the opportunity for relatively better scalability. The Paragon in general, however, shows modest scalabiltiy. Communication cost was observed from the measurements to be the limiting factor still. This can be also noted from figures 5 through 7. With the increase in communications requirements, due to the increase in the levels of decomposition, the speedup curve continues to drop, with best results seen at one level of decomposition and worst at 4 levels.

5.3 Comparative Results

	F8/L1	F4/L2	F2/L4
MasPar MP-2 (16K)	.0169	.0138	.0123
Intel Paragon 1 Proc.	4.227	3.45	2.78
32 Proc.	.613	.632	.6623
DEC 5000 Workstation	5.47	4.54	4.11

Comparative Wavelet Decomposition Performance Measurements
Table 1

Table 1, lists the key measurements to help comparing the performance of the mentioned machines. From the table, it is clear that for the machine sizes and configurations used the MasPar is still favorably performing. This is consistent with SIMD machines that have been know to perform well in image processing applications. The MasPar, with the given configuration, is capable of processing 30 images or more per second. Thus for real-time video, multimedia applications, and scientific and medical applications high-performance computing is quickly asserting its presence.

6. Conclusion

In this study, we have mapped the multi-resolution Wavelet algorithm, developed by Mallat [Mal89], onto the high-performance parallel computers and applied it to remotely sensed data from NASA's Landsat-Thematic Mapper images. The MasPar has provided two orders of magnitude improvement over a workstation, for the specific hardware described here. The Intel Paragon exhibited one order of magnitude improvement and required knowledge about the network operation and special effort to scale beyond 4 processors.

References

[Bla90] Tom Blank, The MasPar MP-1 Architecture, *Proc. of the IEEE Compcon*, Feb. 1990.

[Chan95] Chan, Chui, LeMoigne, Lee, Liu, and El-Ghazawi, The Performance Impact of Data Placement for Wavelet Decomposition of 2-D Image Data on SIMD Machines. Frontiers '95, McLean, VA.

[Chu92] C.K. Chui, *An Introduction to Wavelets*, Wavelet Analysis and its Applications, Volume 1, Academic Press, 1992.

[Cod92] M.A.Cody, The Fast Wavelet Transform, *Dr.Dobb's Journal,* April 1992.

[Dja92] J.P. Djamdji, A. Bijaoui and R. Maniere, Geometrical Registration of Images: The Multiresolution Approach, *Journal of Photogrammetry and Remote Sensing*, Vol. 59, No.5, May 1993, 645-653.

[Dau92] I. Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics, 1992.

[El-Ghaz94] Tarek El-Ghazawi and Jacqueline Lemoigne, Multiresoultion Wavelet Decomposition on the MasPar Massively Parallel System. International Journal of Computers and Their Applications, September 1994.

[Fan89] Z. Fang, X. Li, and L. M. Ni, On the Communication Complexity of Generalized 2-D Convolution on Processor Arrays, IEEETC, February 1989.

[Fan86] Z. Fang, X. Li, and L. M. Ni, Parallel Algorithms for 2-D convolution, *Proceedings of ICPP'86.*

[Lee87] S.-Y. Lee and J. K. Aggarwal, Parallel 2-D Convolution on a Mesh Connected Processor Array, *IEEETPAMI*, July 1987.

[Lee94] H.J. Lee, J.C. Liu, A.K. Chan, and C.K. Chui, Parallel Implementation of Wavelet Decomposition/Reconstruction Algorithms, *Proceedings SPIE Wavelets'94,* Orlando, April 5-8, 1994.

[Lem94] J. LeMoigne, Parallel Registration of Multi-Sensor Remotely Sensed Imagery Using Wavelet Coefficients, *Proceedings SPIE Wavelets'94,* Orlando, April 5-8, 1994.

[Lu93] J. Lu, Parallelizing Mallat Algorithm for 2-D Wavelet Transforms, *Information Processing Letters*, 45, 255-259, 1993.

[Mas92] *MasPar Technical Summary.* MasPar Corporation, November 1992.

[Mal89] S.G. Mallat, A Theory for Multiresolution Signal Decomposition: The Wavelet Representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, July 1989.

[Mar89] M. Maresca and H. Li, Morphological Operations on Mesh Connected Architectures: A Generalized Convolution Algorithm, *Proc. of the 1986 IEEE CS Conf. on Computer Vision and Pattern Recognition,* pp 299-304.

[Num92]*Numerical Recipes,* Chapter 13.10 Wavelet Transforms, pp 591-606, 1992.

[Nic90] J. Nickolls, The Design of the MasPar MP-1: A Cost Efficient Massively Parallel Computer, *Proc. of the IEEE Compcon*, Feb. 1990.

[Str89] G. Strang, Wavelets and Dilation Equations: A Brief Introduction, *SIAM Review,* Vol. 31, No. 4, pp. 614-627, December 1989.

[Sto83] Q. F. Stout, Mesh-Connected Computers with Broadcasting, *IEEETC*, September 1983.

APPENDIX B:

ASTROPHYCIAL SIMULATIONS ON HIGH-PERFORMANCE COMPUTERS

In Collaboration with A. Meajil (GWU) and A. Ozkaya (GWU)

An Empirical Study of Parallel Overhead and Scalability of Scientific Simulations

Abstract

High-Performance Computing Systems based on parallel processing have the potential for satisfying the rapid growth in computational requirements of large physical simulation problems, at economical costs. As such machines grow in size, however, parallelization overhead grows in a manner that can limit scalability. This work sheds some light on the sources, dynamics, and magnitudes of the different types of overhead and their impact on performance. Results are obtained through experimental measurements of NASA Earth and Space Sciences (ESS) astrophysical simulations, running on the JPL/ESS Intel Paragon and CRAY T3D.

1. INTRODUCTION

Experimental HPC systems based on parallel processing architectures offer the opportunity to achieve orders of magnitude performance gain for existing problems as well as make feasible the solution of problems of much greater size and resolution. Information derived from evaluation studies can enhance our understanding of the potential growth path of high-performance computing and reveal possible difficulties that inhibit advances. The Joint NSF/NASA Initiative on Evaluation (JNNIE), is a national evaluation activity which involves NSF and NASA centers and, thus, includes a large number of testbeds and applications. JNNIE is concerned with such issues as application characterization, usability, macro performance, and micro performance. In this work, which is conducted in the framework of JNNIE, micro-performance only is considered. Micro-performance is concerned with metrics and structured evaluation methods to discover the sources of performance degradation in the basic observable behavior of a machine, against an imaginary ideal. Examples of these sources could be starvation, latency, contention, and overhead. Micro-performance measurements collected here have focused on interprocessor communications overhead, redundancy overhead, load-imbalance overhead, and time spent doing useful work. We refer to this set of measurable quantities as the performance budget. The target applications were all selected from NASA Earth and Space Sciences (ESS) domain. The measurements presented here are from two specific applications: (1) the N-body simulations, and the (2) Particle-In-Cell (PIC) simulations. The target high-performance computing platform for this study were the ESS Intel Paragon and CRAY T3D at the Jet Propulsion Laboratory (JPL).

This paper is organized as follows. Section 2 discusses the selected NASA ESS applications as well as the target high performance computing platforms for this study. Section 3 discusses the types of overhead measurements collected. Section 4 presents the measurements collected for the N-body and the PIC simulations, respectively. Observations and conclusions are derived in Section 5.

2. APPLICATIONS AND SYSTEM SCOPE

As mentioned earlier, the NASA Earth and Space Sciences (ESS) applications considered in this work are the N-body simulations and the Particle-in-Cell (PIC) simulations. The classical N-body problem simulates the evolution of a system comprising n bodies (particles), under the influence of forces exerted on each body by the whole system. Typical domains of application include (i) astrophysics, where the bodies can be viewed as stars or planets in a galaxy, (ii) molecular dynamics, where the bodies are molecules or atoms, and (iii) plasma physics, where the bodies are ions or electrons. The example problem we use in this paper is a simulation of interacting galaxies from astrophysics. The problem studies how the positions and velocities of stars in the galaxies evolve with time under the gravitational forces that the stars exert on one another.

Numerical Particle-Mesh techniques, also known as Particle-In-Cell (PIC), are commonly used to model plasmas, gravitational N-body systems, and both compressible and incompressible fluids [3,9,12]. They represent a popular variant on particle simulation techniques utilizing a numerical grid to more effectively compute the forces acting on the particles. The naive particle-particle approach is only useful in modeling a system with a small number of particles (<10000) because of the very rapidly growing computational complexity. Compared to the most effective solutions, particle-mesh techniques work better when the particle density distribution is relatively uniform. Tree codes, on the other hand, are favored in systems with large density contrast.

These two applications were ported to the target platforms and measurements were collected to study overhead and scalability.

2.1 The Target Systems

The JPL/ESS Paragon and T3D were used to conduct this study. The Paragon has a total of 64 nodes organized into a 16x4 mesh of which 54 are compute nodes and 8 are service nodes. Each node, an Intel GP node, is essentially a separate computer, with one compute and one communication i860 processors. Each of the 56 compute nodes has 32 MBytes of memory. The service nodes include: 4 I/O nodes with 32 MBytes memory and a 4.8 Gbyte RAID each, 1 HIPPI node with 32 MBytes memory, 1 User Service node with 32 MBytes memory, and 1 boot node with 32 MBytes memory and a 4.8 Gbyte RAID. The peak performance (using 56 nodes) is 5.6 GFlops in single precision with an aggregate memory space of 1.8 GBytes and aggregate online disk capacity in excess of 20 GBytes. The programs can be developed in C or FORTRAN which are supported by NX library routines for communication and synchronization purposes.

The Cray T3D is a MIMD system with physically distributed but globally addressed memory. The JPL T3D has a Cray Y-MP as its host system and currently consists of 256 processors each with 2 MWords (16 MB) of DRAM memory. About 25% of the memory is required by the UNICOS microkernel, therefore, the users can expect to have 12 MB of memory for program and data. Each PE is a 64-bit DEC Alpha microprocessor with a frequency of 150 Mhz capable of achieving 150 MFLOPS. The memory interface between the processor and the local memory extends the local virtual address space to a global adderess space. The Alpha processor has a direct-mapped data cache organized into 256 lines with 32 bytes per line. Programs can invalidate the local cache as needed to maintain the coherencey. Also, remote data entering a processor's local memory can invalidate the corresponding cache line. The system is space-shared into partitions where the numbers of processors are powers of two. A node consists of two processors sharing a network support logic. All processors are connected by a bi-directional 3-D torus system interconnect network. This topology ensures short connection paths and high bisectional bandwidth. Channels between nodes are two bytes wide and the peak interprocessor communication rate is 300 MB/sec. in every direction through the torus. The system software includes FORTRAN (a superset of FORTRAN 77 including many FORTRAN 90 array syntax statements), C, and C++ compilers as well as tools for application performance analysis and parallel code debugging. The PVM is currently supported as are some lower level Cray libraries for passing data and messages among processors.

In order to allow accurate measurements of communications, the message passing programming model was used. All applications were developed in C and augmented with the appropriate NX or PVM communication calls. The applications used the "single program, multiple data" (SPMD) programming model. In this model, the same program runs on each node in the application, but each node works on a part of the data. However, because each node is an independent computer, one can also use other programming models. One example is the "manager-worker" model, in which a "manager" program starts up several "worker" programs on other nodes, then gathers and interprets their results.

In our implementations, the PIC application used a worker-worker SPMD model, while the N-body used the manager-worker model. With this model, the manager creates the tree where all spatial information about all particles are inserted. Then, the manager broadcasts the tree to all nodes. Each node manipulates only a subset of the particles in order to compute the essential forces that interact with those particles. Therefore, each node does its work without access to data that is being held by other nodes. Each "worker" node updates the information of all of its particles. The "worker" node, then, sends its updated particles to the "manager" node in order to create an updated tree which is to be used in the next time-step.

2.2 N-body Problem and the Barnes-Hut Method

The general N-body problem may be stated as the following set of ordinary differential equations [15]:

$$dx_i / dt = v_i \qquad (1)$$

$$m_i \, dv_i / dt = \Sigma_{j \neq i} F_{ij} \qquad (2)$$

In astrophysical simulations, the force term, F_{ij} is the Newtonian gravity:

$$F_{ij} = \frac{G m_i m_j \, r_{ij}}{} \qquad (3)$$

$|r_{ij}|^3$

The gravitational force is "long-range", meaning that there is no cutoff point, beyond which the force may be considered negligible. In principle, it is necessary to evaluate the entire sum on the right-hand side of (2) at each time step of the time integration. Naively, this requires $O(N^2)$ operations at each time step.

The Barnes-Hut (BH) [2] algorithm is one of a number of algorithms [1,2,8,10,11] that use a multiple expansion and a hierarchical data structure to reduce the complexity of computing long-range interactions like gravity. The multipole expansion allows one to treat a collection of bodies as a point mass (perhaps with quadrupole and higher moments) located at the center of mass. In Figure 1, the force on point x_i may be evaluated approximately as:

$$F_i = \Sigma_j \frac{G m_i m_j\, r_{ij}}{|r_{ij}|^3} \approx \frac{G m_i M R_{cm}}{|R_{cm}|^3} \qquad (4)$$

The quality of the approximation in (4) is a decreasing function of the ratio: $b/|R_{cm}|$, where b is the radius of the collection of bodies. In the BH algorithm, multipole moments are computed for cubical cells for an octtree of variable depth. The tree is constructed at each time step with the following properties:

1. The root cell encloses all of the bodies.
2. No terminal cell contains more than m bodies.
3. Any cell with m or fewer bodies is a terminal cell.

A typical two-dimensional BH tree with m=1 is shown in Figure 2.

To compute the force on a body, one traverses the tree starting at the root. Any time a cell with a sufficiently small value of $b/|R_{cm}|$ is encountered, the multipole approximation is utilized. Thus, distant cells, which comprise many individual bodies, may

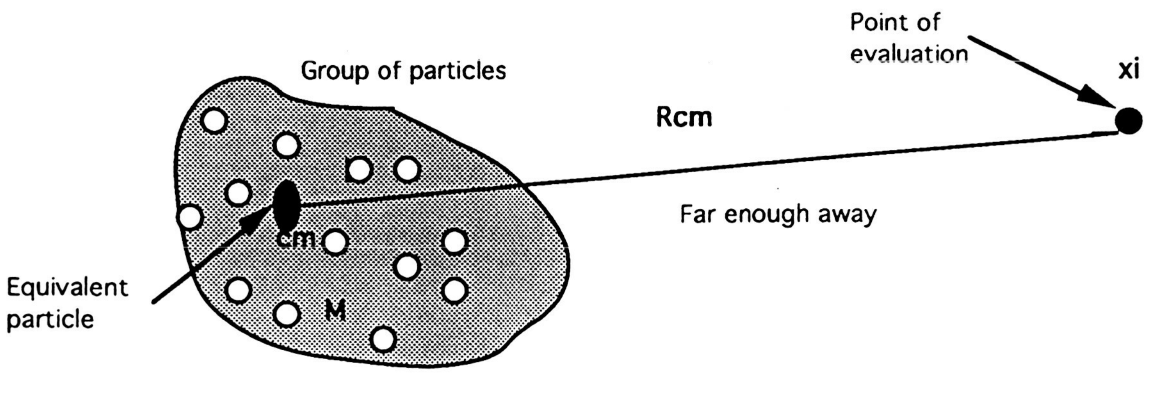


Figure 1. Approximation of particles by a single point mass.

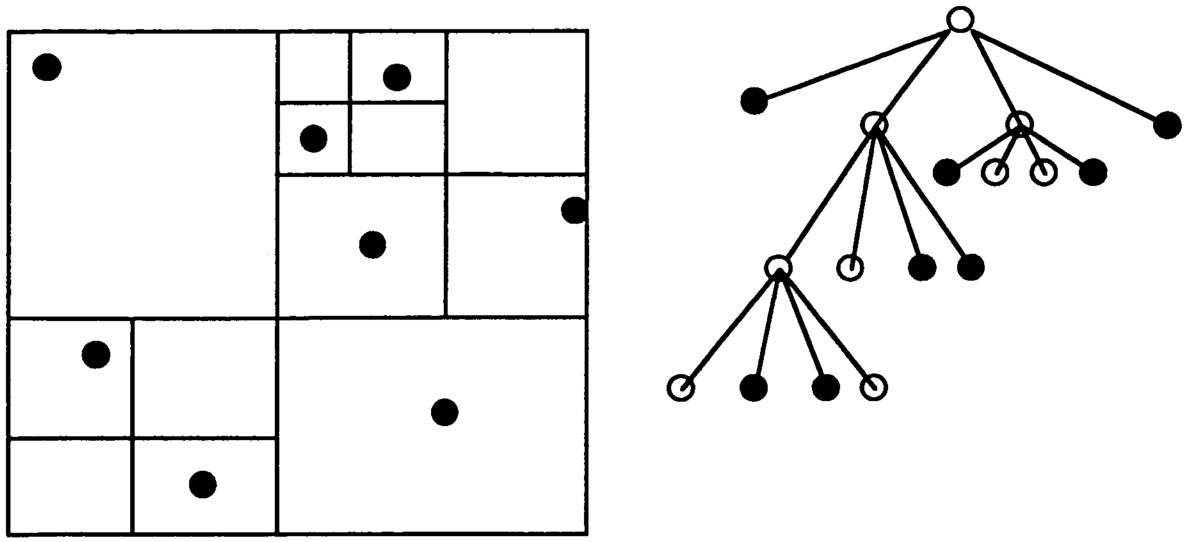


Figure 2. A 2-D particle distribution and its quadtree.

be approximated in unit time. The resulting algorithm, when applied to all bodies, requires O(N.logN) operations to evaluate the forces on all N bodies.

In the BH method, the force-computation phase within a time-step is expanded into three phases:

1. Building the tree: The current positions of the particles are first used to determine the dimensions of the root cell of the tree. The tree is then built by adding particles one by one into the initially empty root cell, and subdividing a cell into its four children as soon as it contains more than a single particle.

2. Computing cell centers of mass: An upward pass is made through the tree starting at the leaves, to compute the center of mass of internal cells from the centers of mass of their children.

3. Computing forces: The force-computation phase consumes well over 90% of the sequential execution time in typical problems, and is described in detail below.

The tree is traversed once per particle to compute the net force acting on that particle. The force-computation algorithm for a particle starts at the root of the tree and conducts the following test recursively for every cell it visits: If the cell's center of mass is far enough away from the particle, the entire subtree under that cell is approximated by a single particle at the cell's center of mass, and the force this center of mass exerts on the particle is computed. If, however, the center of mass is not far enough away from the particle, the cell must be "opened: and each of its subcells visited.

4. Updating particle properties: Finally, the force acting on a particle is used to update such particle properties as acceleration, velocity and position. This phase does a constant amount of work per particle, so its computational complexity is O(N).

Conceptually, the main data structure in the application is the Barnes-Hut tree. Since the tree changes every time-step, it is implemented in the program with two arrays: an array of bodies that are leaves of the tree, and an array of internal cells in the tree. Among other information, every cell has pointers to its children, and it is these pointers that maintain the current structure of the tree. The structure representing a body holds 56 bytes of data in two dimensions. There is also a separate array of pointers to bodies and one of pointers to cells. These arrays are used by the processors to determine which bodies and cells they own. Every processor owns an equal contiguous chunk of pointers in these arrays, and each chunk is larger than the maximum number of bodies or cells a processor is expected to own. The total data space of the program is linearly proportional to the number of bodies for both uniform distributions (balanced tree) and non uniform ones.

The domain decomposition technique used here is Costzones partitioning [16]. In this method, the summation of works associated with all particles is used to divide the workload equally among the processors. This technique is very simple and does not have much computational overhead associated with it, when compared with other popular methods, such as the Orthogonal Recursive Bisection (ORB)[15].

The Costzones technique takes advantage of another key insight into the hierarchical methods for classical N-body problems, which is that they already have a representation of the spatial distribution encoded in the tree data structure. Consequently, one can partition the tree rather than partitioning the space directly. In the Costzones scheme, the tree cell's children laid out from left to right in increasing order of child number. The cost of every particle, which is the total amount of interactions between the particle and all others, as counted in the previous time step, is stored with the particle. Every internal cell holds the sum of the costs of all particles that are contained within.

The total cost in the domain is divided among processors so that every processor has a contiguous, equal range or zone of costs (hence the name Costzones). For example, a total cost of 1000 interactions would be split among 10 processors so that the zone comprising costs 1-100 is assigned to the first processor, zone 101-200 to the second, and so on. Which costzone a particle belongs to, is determined by the total cost up to that particle in an inorder traversal of the tree.

In our implementation, building the tree is done at a manager node. After building the BH tree, the manager broadcasts the BH tree to the worker nodes. Therefore, an identical copy of the BH tree would be available in each processor.

Since each processor has a subset of the bodies and a whole copy of the BH tree, there is no more need for interprocessor communication. In fact, the original serial code for force evaluation may be used completely unchanged. The parallel algorithm will produce results identical to the serial algorithm, except for a very small amount of roundoff error which results from the non-associativity of floating point operations.

2.3 The PIC Problem

In the PIC approach, the force equation is based on continuum representations of the charge density and electric field. Poisson's equation

$$^2 \phi = -\rho(x) \qquad (1)$$

relates the charge density $\rho(x)$ to the electric potential ϕ. The electric field is

$$E(x) = - \quad \phi \qquad (2)$$

and the force on particle i is

$$F_i = q_i \; E(x_i) \qquad (3)$$

where q_i is the charge on particle i and x_i is the location of it. To solve these equations, finite difference approximations on a grid are used in the following four steps:

1) Calculate ρ at each grid point based on the positions of particles. In this step, a Cloud-In-Cell charge assignment scheme is employed. If the particle position x_i is located between the two grid cell centers x_{g-1} and x_g, the charge assigned to each of these cells is given by

$$\rho_g = q_i \; (x_i - x_{g-1}) \;/\; \Delta x \quad \text{and} \quad \rho_{g-1} = q_i \; (x_g - x_i) \;/\; \Delta x$$

where Δx is the grid cell size.

This operation scales as $O(N_p)$, the number of particles.

2) Solve (1) using FFT. This scales as $O(N_g . \log N_g)$, where Ng is the number of grid points; then evaluate E at each grid point using (2), which scales as $O(N_g)$.

3) Use interpolation and (3) to evaluate the force on each particle. In this step, the finite difference approximation

$$E_g = - (\phi_{g+1} - \phi_{g-1}) / 2\Delta x \qquad \text{is used.}$$

This step scales as Np.

4) Solve particle equations of motion, that are,

$$dx_i / dt = v_i \qquad \text{and} \qquad dv_i / dt = F_i / m_i$$

where v_i and m_i are the velocity and mass of particle i, respectively, to find the new locations and velocities of the particles. This scales as $O(N_p)$.

Combining these steps, the whole scheme has the complexity of $O(N_p + N_g . \log N_g)$.

In general, these four steps involve computation and communication between the data structures representing the field data and the particle data. The field data is represented as an array where domain neighborhood relationships are maintained by the indices of the array cells. The particle data array, however, maintains no special relationship to neighboring elements. Domain decomposition for PIC codes [4, 5, 12] can be used for both particles and fields. The load on any processor has the components of particle related and grid related operations. The objective should be to adjust the domain boundaries so that the total load is distributed over the processors evenly. It is possible therefore that one processor might spend more of its time on particles and less on grid points than another processor, and yet both would complete at the same time. This benefit cannot be realized for electrostatic codes because the global character of Poisson's equation forces the processors to be synchronized both at the start and the end of the field solver equation.

The developed parallel code is an implementation of 3-D Electrostatic PIC Simulation. The particles are finite sized charge clouds which are divided among the processors uniformly. The particle data structures are vectors holding the position and velocity information. The field data structures are 3-D arrays with wrap-around boundary conditions. The algorithm includes an adaptive time-step adjustment scheme in order to prevent the particles from moving any further than neighboring grid cells. Each processor processes its own particle data when charging corresponding grid points. These charges are collected from the processors with a global summation operation. A 3-D parallel FFT routine, realized using slab decomposition and a Paragon 1-D FFT library routine, calculates the potentials of the grid locations from grid charges. Note that right before and after this field solving process, every processor will have the global charge and field information, respectively. This type of decomposition results in a parallel execution time with an input size dependent component and a constant one which is related to the grid size, should the grid remain the same for simulations of different number of particles. For any realistic simulation, the number of grid cells is much smaller than that of the particles. For the 3-D FFT, the data are stored in such a way that each "plane" formed by two of the dimensions is entirely within one processor and the other dimension is divided among the processors. The 3-D transform is performed by factoring it into 1-D transforms along each of the dimensions. Thus, two of these dimensions can be processed by transforms within single processors, without communication. To transform along the other dimension within single processors, the data are rearranged among the processors so that the slabs contain this third dimension, and one of the other dimensions is now divided among processors. Therefore, the decomposition of data with regard to the dimensions is different in the space and frequency domains. At the end, every processor will have a slab of the electric potential data which must be made global for electric field calculations.

3. PERFORMANCE BUDGET

The intended performance measurements are designed to correlate the system scalability to parallel overhead, from the user/application perspective. Therefore, our performance budget model relies on application instrumentation and breaks the overall parallel execution session into non-overlapping useful processing time and a number of overhead components. Desirable architectural features, such as the ability to hide latency, as well as good parallel programming practices, such as the use of asynchronous rather than synchronous communications, are therefore favored by this model. The types of overhead identified here are the average communication overhead, imbalance overhead, and redundancy overhead. Each of these types of overhead is reported as a percentage of the parallel execution time. The general programming model used was, again, the SPMD model due to its popularity. To facilitate the communications overhead measurements, the code was developed using the message-passing paradigm.

The communications overhead is measured directly and averaged over all processors used for the computation. A communication transaction is measured from the point of initiating the communication system call, till the call returns. Redundancy overhead refers to the additional operations needed to facilitate the parallelization. Two sources of redundancy are differentiated here. The first is the parallel duplication, in which the same operation is duplicated using the same data values at all processors. The second is the unique redundancy, in which different or similar but not identical processing is done to allow the parallelization. Example of the duplication redundancy is the initialization of a loop counter by the same value at all processors. Example of the unique parallelization redundancy is operations that pertain to domain decomposition, where each processor tries to figure out which part of the data it will be working on. Given n processors, (n-1) copies of the duplication redundancy and all unique redundancy are averaged over all processors to produce the redundancy overhead. Imbalance/wait overhead is the time difference between the max completion time and the minimum completion time over all processors. All timing measurements were wall-clock timing and timed activities were selected such that no global knowledge of time was needed.

4. EXPERIMENTAL MEASUREMENTS ON THE PARAGON AND T3D

Measurements of instruction mixes and serial execution were collected as a first attempt to understand the characteristics of the underlying simulation codes and the consequent serial execution behaviors. The Nbody was foud to have a higer percentatge of integer operations (almost 60%) due to the tree construction and manipulations, compared to only 32% in the case of PIC. PIC, however, was characterized with high memory requirements (almost 40% of the instructions are load/store) due to comparisons with all particles, and higher floating point operations (23%) mostly due to field calculations. In Nbody, 26% of the instructions only were load/store and 14% were floating point operations.

Table 1 and Table 2 show how such instruction mixes tarnslated into differences in the execution times trends. For example, moving from the Paragon (i860 processors) to the T3D (DEC Alphas), PIC shows a little improvement in speed, while the Nbody, with its dominant integer manipulations and less memory, is showing up to one order of magnitude improvement.

Table1:
Sample Serial Execution Times per Iteration on the Paragon

PIC:

Size	time(m=32)	time(m=64)
256K	13.35 sec.	21.92 sec.
512K	24.41	34.85
1M (extrapolated)	45.93	58.31
1M (real)	249.20	820.41

N-body:

Size	time
1K	5.77 sec.
8K	53.27
32K	237.51

Table 2:
Sample Serial Execution Times per Iteration on the T3D

PIC:

Size	time(m=32)	time(m=64)
256K	5.53 sec.	17.02 sec.
512K	9.74	21.17
1M	18.34	29.49
2M (extrapolated)	35.18	46.12

N-body:

Size	time
1K	0.53 sec.
8K	6.31
32K	30.90

4.2 Paragon Measurements
4.2.1 N-Body Measurements

A number of experiments were conducted to reveal the behavior of the N-body application with respect to scalability and overhead. These were performed for different input data sizes as well as for different number of processors. Figure 3 summarizes the scalability measurements. In this figure, N-body scales nicely with the increasing number of processors, particularly when large data sets are used. This is consistent with the intuition driven from the way the parallel program works. In the used parallel program, building the tree was done sequentially at the manager node, recall the manager-worker model. This sequential part requires traversing the tree only once. On the other hand, computing the forces at each body was parallelized, which requires N traverses of the tree in the sequential case. With such N:1 growth ratio in the parallel to the sequential parts, near-linear speed up is expected. However, due to the rising communications cost associated with the centeralized construction of the tree, as the size of the machine grows and processors become more distant from one another, a gradual drop in efficiency is observed.

Figures 4 through 6 report the overhead measurements for 1K, 4K, and 32K bodies, respectively. As the number of processors increase, a corresponding increase of communications overhead and imbalance take place (Figure-4). While the costzone decomposition guarantees equal computational effort by all processors, the imbalance overhead continues to increase as more processors are used. This is a side effect due to the use of the manager-worker model, as distance variability from the manager increases with the increased number of workers. However, as the input data size gets larger, most of this overhead is amortized nicely, as shown in Figure 5&6. This is due to the rapid growth in the parallel part of the program, as pointed before. Redundancy overhead, on the other hand, has been minimal in all cases.

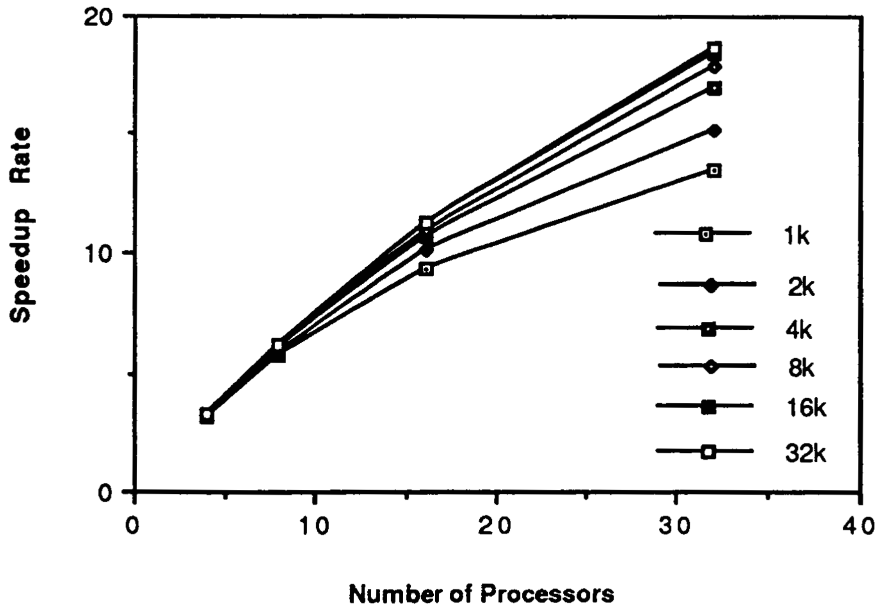


Figure 3. Scalability of the N-body on the Paragon.

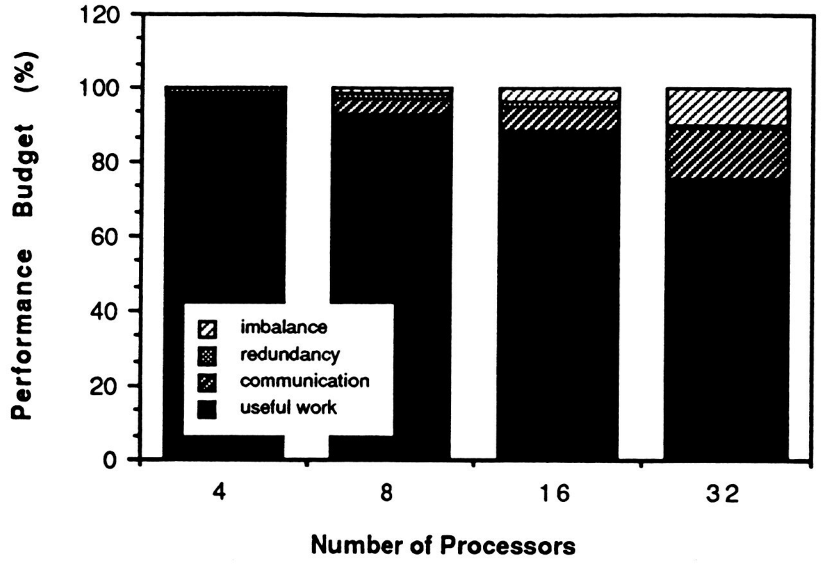


Figure 4. N-body Performance Budget at 1K Bodies.

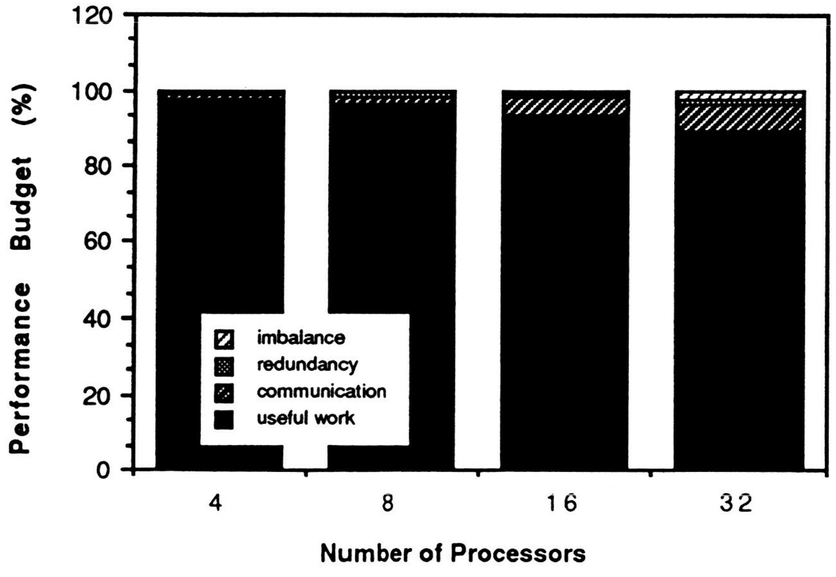


Figure 5. N-body Performance Budget at 4K Bodies.

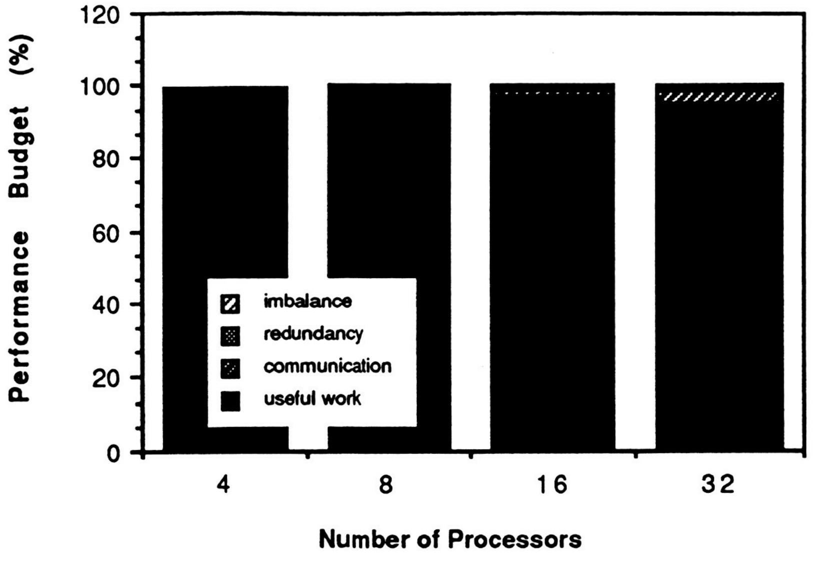


Figure 6. N-body Performance Budget at 32K Bodies.

4.2.2 PIC Measurements on the Paragon

PIC measurements are summarized in Figure-7 through Figure-14. As one can vary both the grid size and the number of particles, measurements are reported for different number of particles and processors, considering two grid sizes, 32x32x32 (m=32) and 64x64x64 (m=64). It should be mentioned that we have initially used the NX routine *gssum* (global sum of floating-point vectors) to make the global charge and electric field information available in every processor. However, since the grid computations require rapidly growing global communication and gssum seems to be implemented using many many-to-many communications, speed up dropped significantly with more than 8 processors. Our measurements have shown that the *gssum* consumes most of the total communication time and does not scale well with the number of processors. It works very efficiently for 4- and 8-processor partitions, but for 16- and 32-processor ones. On the other hand, it has been observed to scale well with the amount of data communicated. To reduce the communication overhead, we have implemented our own global sum routine based on parallel-prefix algorithm using many one-to-one communications. As shown in Figures 7 and 8, the resulting code exhibits good scalability, which becomes better as the simulation size is increased because of better amortized communication overhead. Also,figure 7 generally exhibits better speedup factor, than that of 8. This is due to the increase of global communications associated with the increased size of the grid.

In Figures 7 and 8, the uniprocessor time was obtained through extrapolation for the cases of 1M and 2M particles, at which excessive paging was observed. This is necessary to reflect realistic projections of speedup, non superlinear, when sufficient memory is available. Superlinear scalability could also occur due to improved caching when data is partitioned over processors in a multiprocessor system. Figure 9, however, was obtained by direct measurements to examine the superlinear effect of paging. In this figure, speedup increases suddenly for simulations that used more than 640K particles. This was due to the fact that excessive paging was occurring when the uniprocessor measurements were for 640K particles or more.

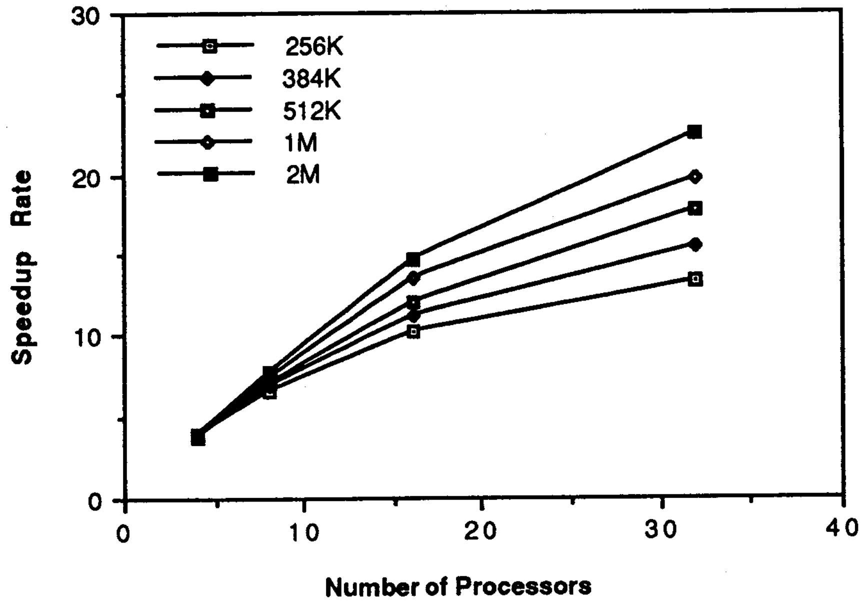


Figure 7. PIC Scalability on the Paragon for a 32x32x32 Grid.

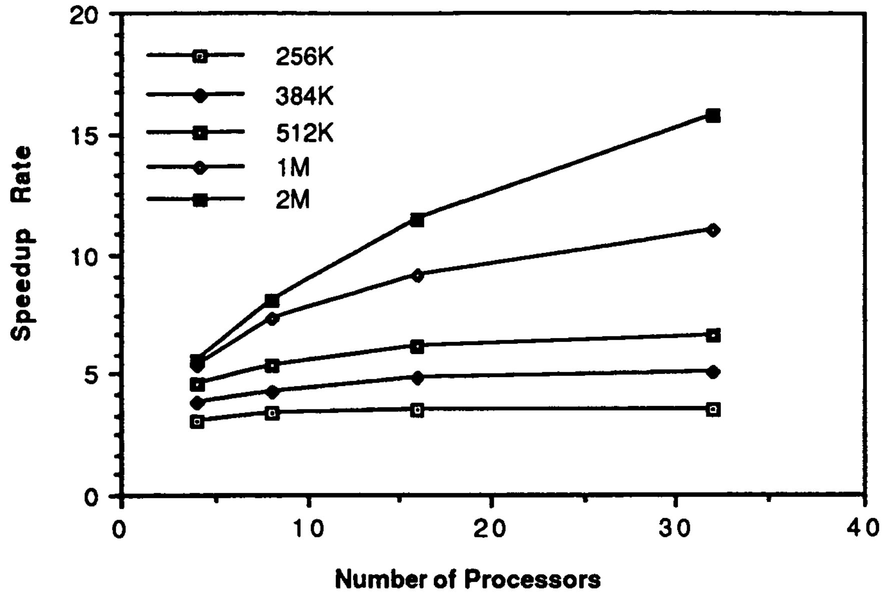


Figure 8. PIC Scalability on the Paragon for a 64x64x64 Grid.

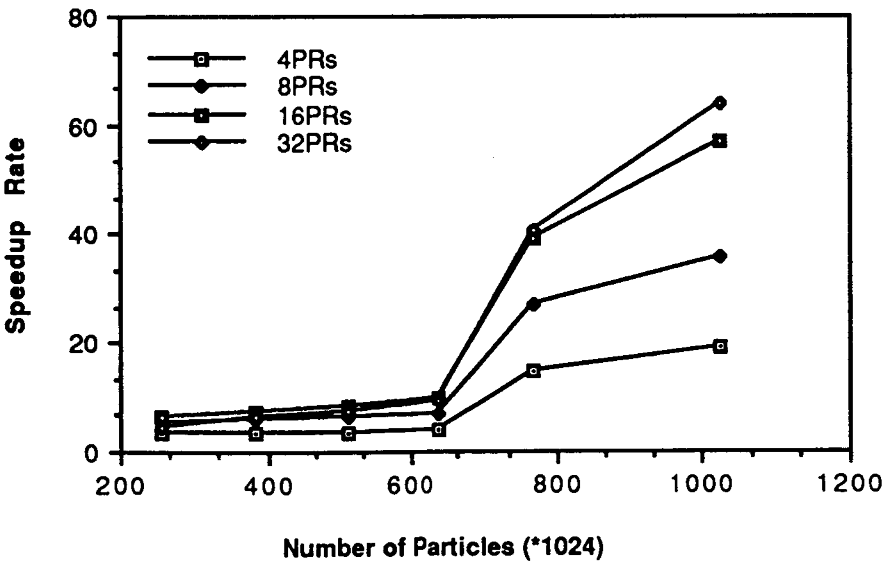


Figure 9. Superlinear Speedup Behaviors (m=32).

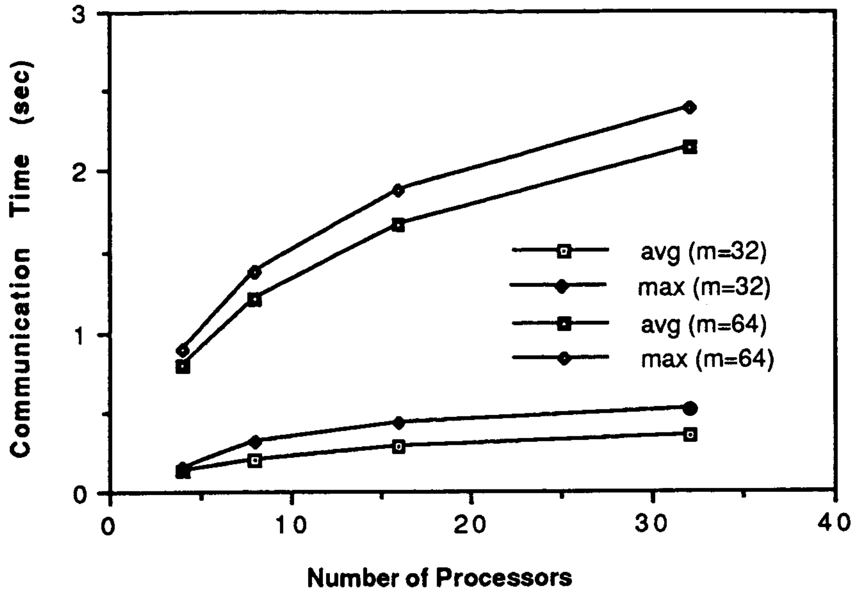


Figure 10. Communication Measurements for PIC.

The communication behavior is emphasized in Figure 10. This figure shows that there is not much difference between average and maximum times spent for communication during each iteration which indicates that communication activities are well balanced, due to the worker-worker model.

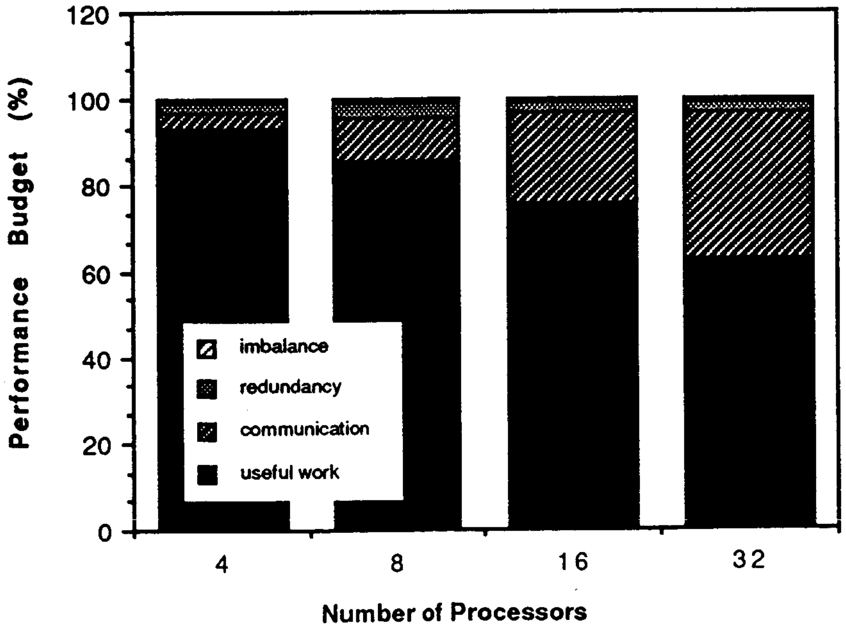


Figure 11. PIC Performance Budget for 256K particles

and m=32.

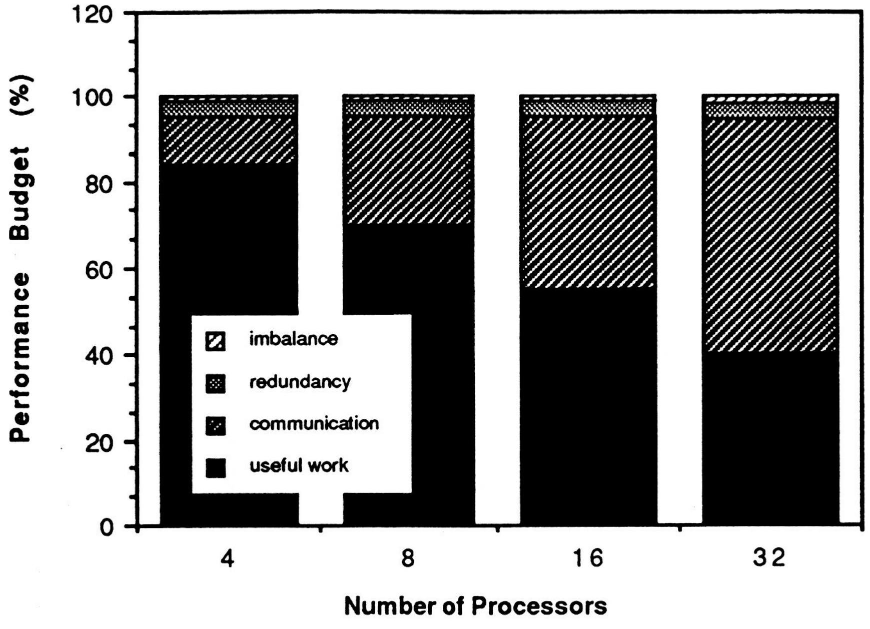


Figure 12. PIC Performance Budget for 2M particles and m=32.

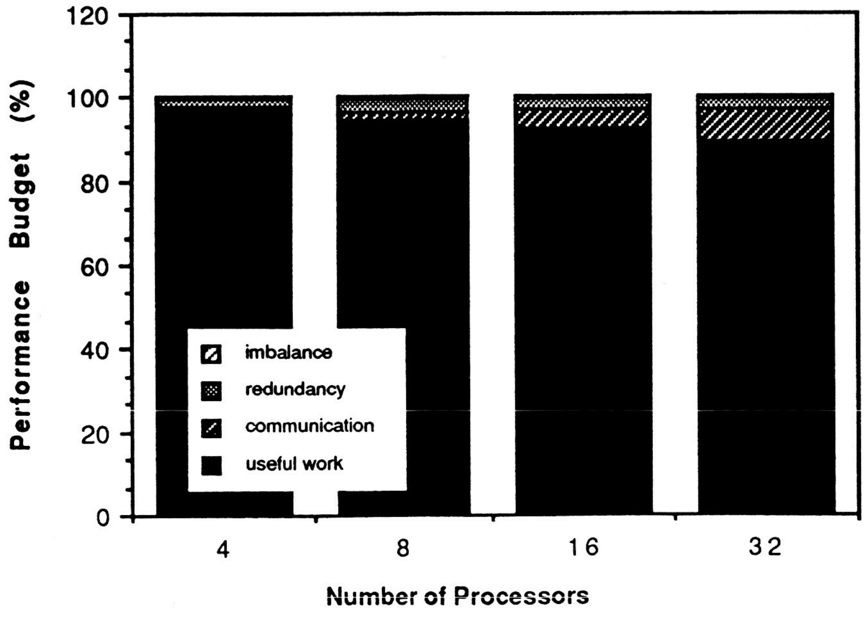


Figure 13. PIC Performance Budget for 256K particles and m=64.

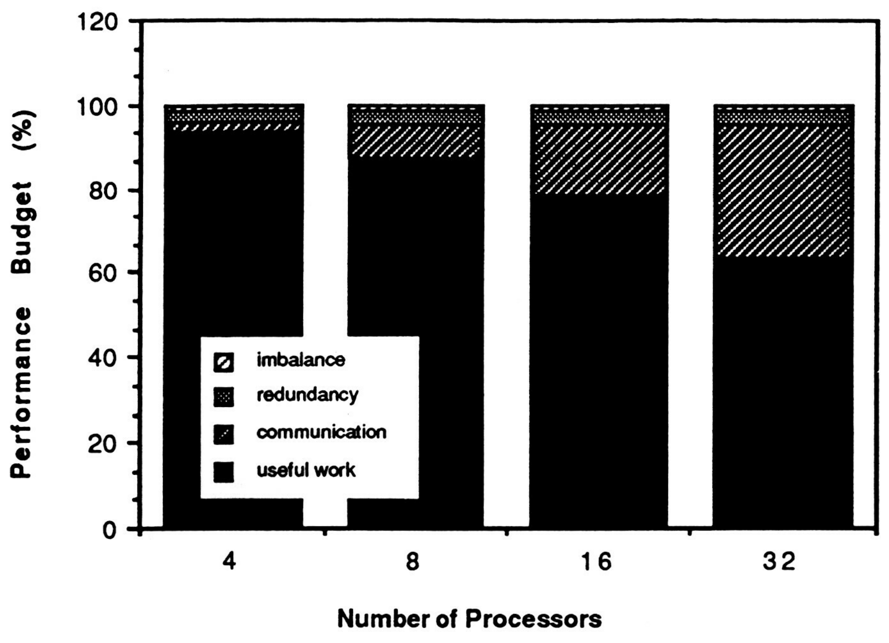


Figure 14. PIC Performance Budget for 2M particles and m=64.

The performance budget graphs, Figures 11 through 14, show that the redundancy component is not substantial. Communication component grows quickly with increasing grid size and becomes the dominant activity when the data size is not large enough. Amortization of overhead can be seen by comparing Figure 11 with Figure 12, and Figure 13 with Figure 14. However, for the bigger grid size, the large increase in communications makes the overhead hard to hide by increasing the data volume 8 folds. Imbalance has been observed to be rather variant, increasing with the amount of communication but is negligibly small.

4.3. MEASUREMENTS ON THE T3D

4.3.1 N-Body Measurements

In figure 15, the scalability of Nbody on the T3D is presented. The smaller communication did not result in better scalability than the case of the Paragon, for 32 processors or less. This is due to the fact that the alpha processor is faster for Nbody, as shown in tables 1 and 2, which makes the computation/communication ratio smaller, although the execution is faster. Figures 16-18, show the performance budget. The ratio of the useful work is again small as compared to the Paragon due to the fast processor.

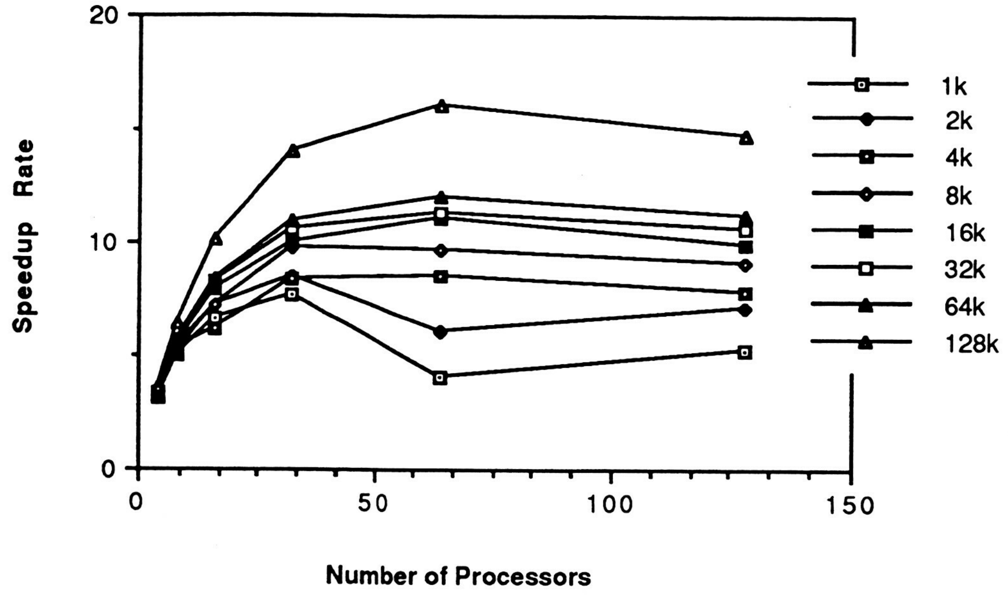


Figure 15. N-Body Scalability

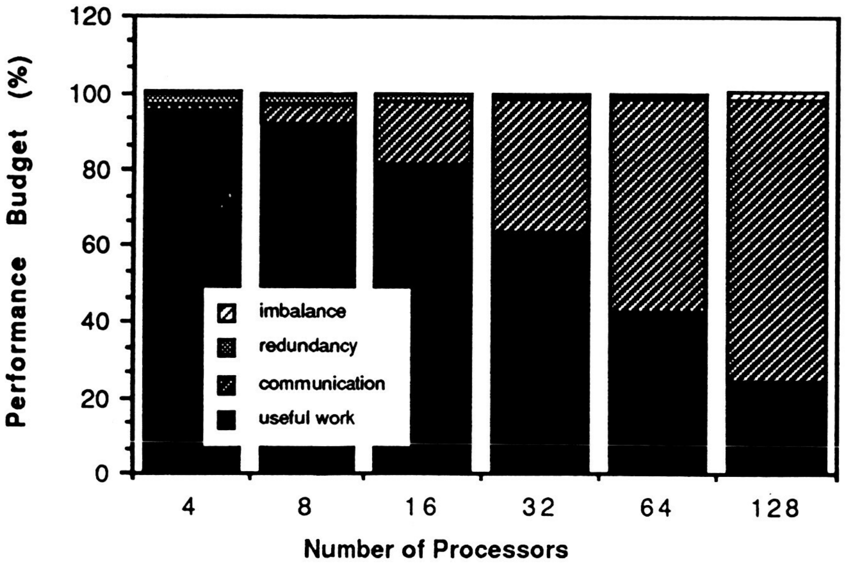


Figure 16. N-Body Performance Budget for 1K particles.

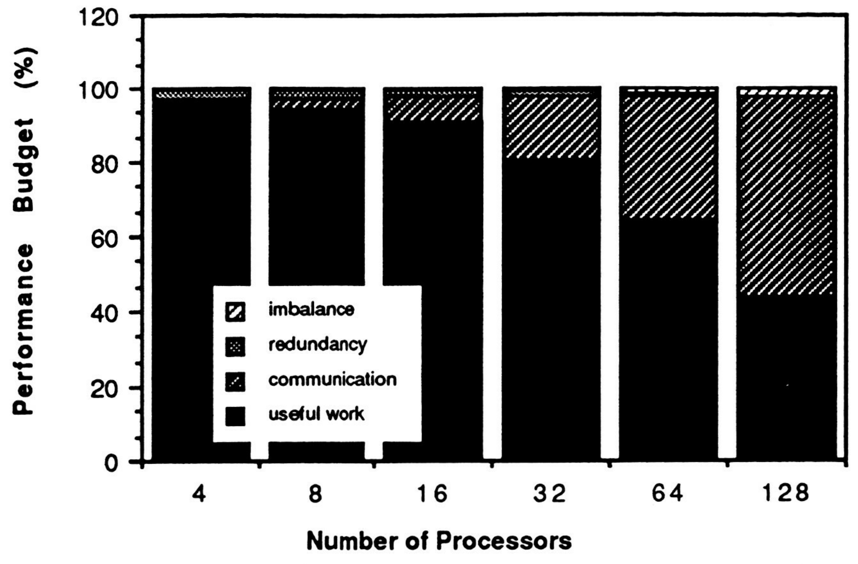


Figure 17. N-Body Performance Budget for 4K particles.

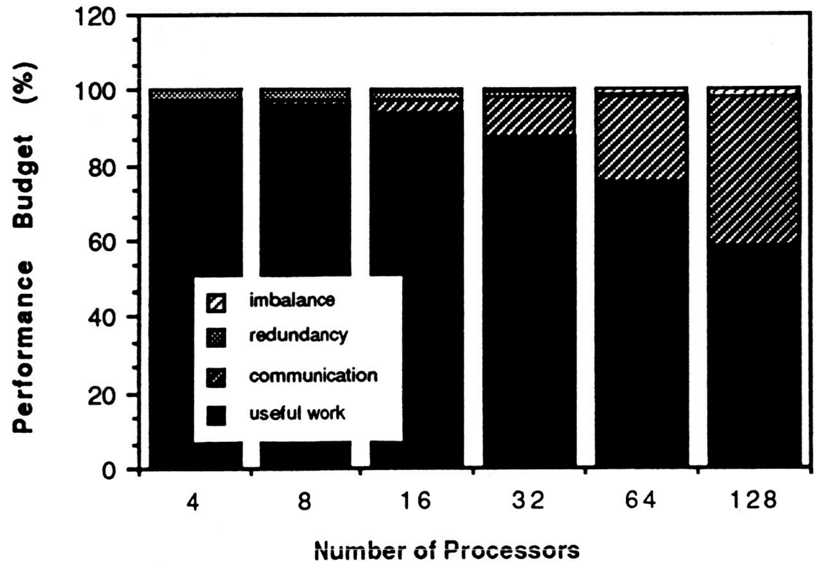


Figure 18. N-Body Performance Budget for 32K particles.

4.3.2 PIC Measurements

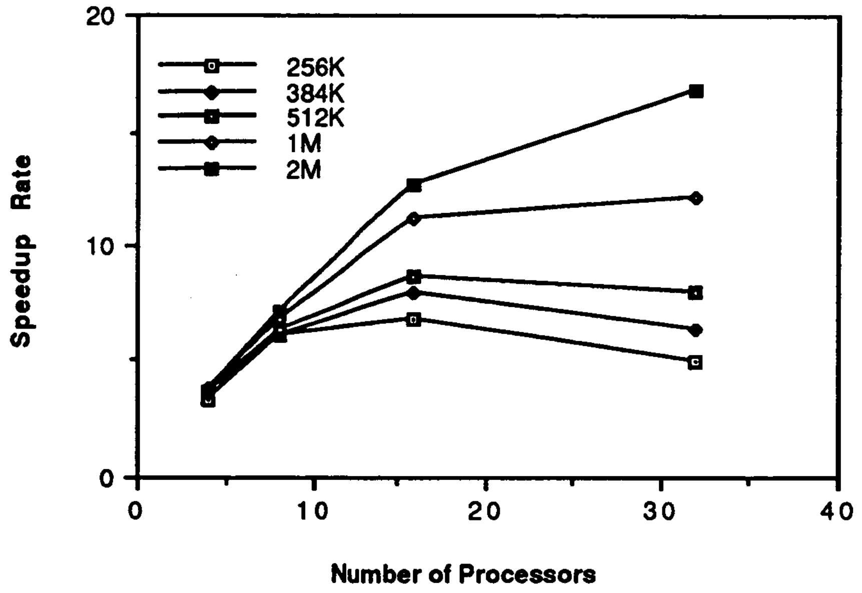


Figure 19. PIC Scalability on the T3D for a 32x32x32 grid.

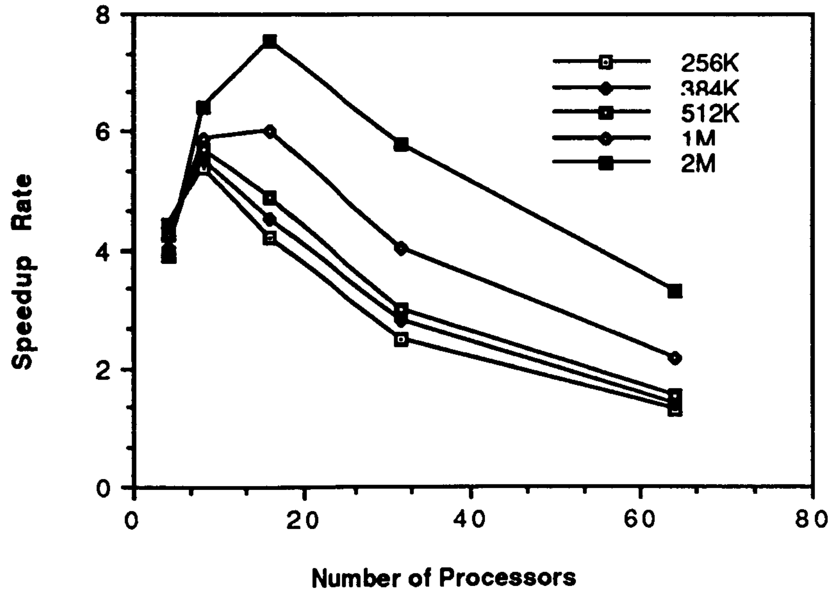


Figure 20. PIC Scalability on the T3D for a 64x64x64 grid.

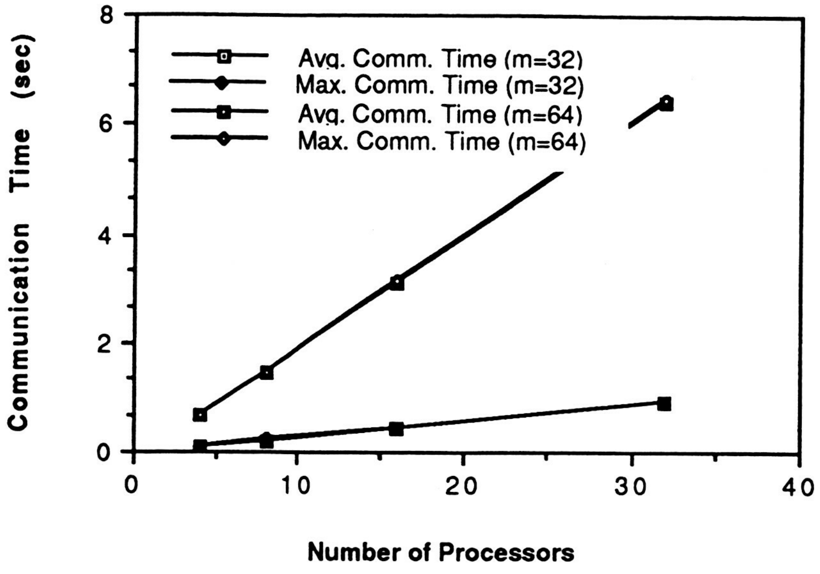


Figure 21. Communication Measurements for PIC on T3D.

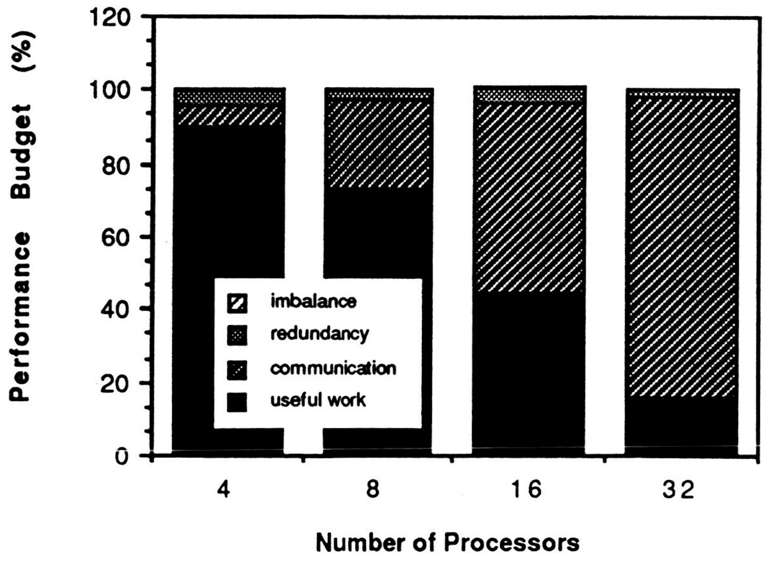


Figure 22. PIC Performance Budget for 256K particles and m=32.

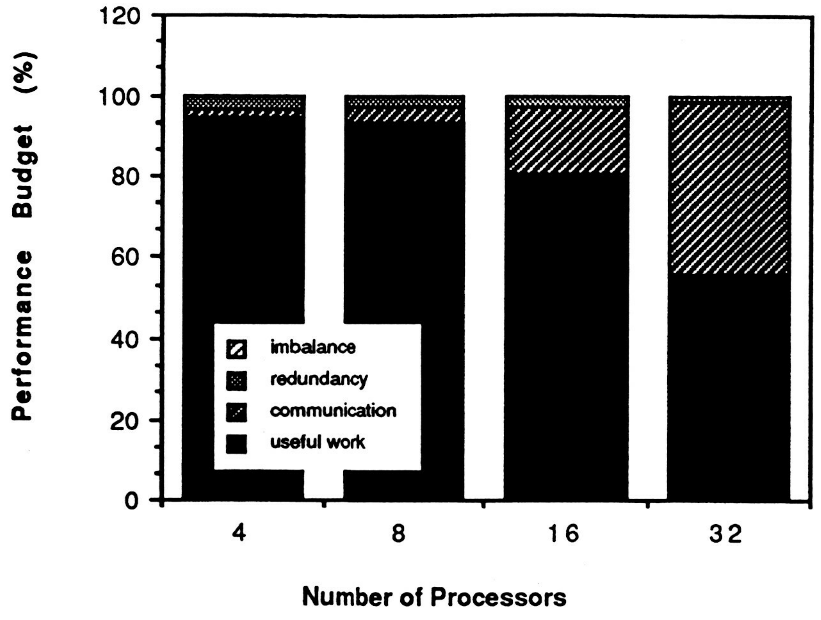


Figure 23. PIC Performance Budget for 2M particles and m=32.

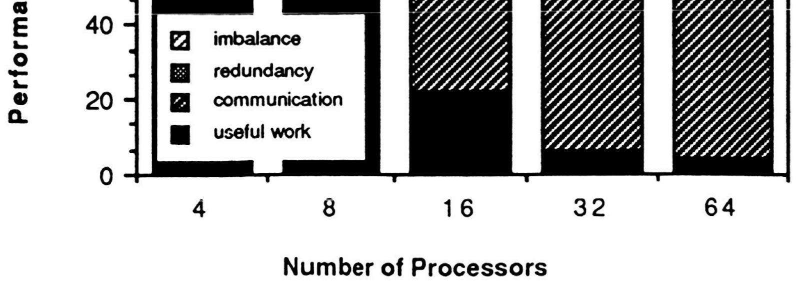


Figure 24. PIC Performance Budget for 256K particles and m=64.

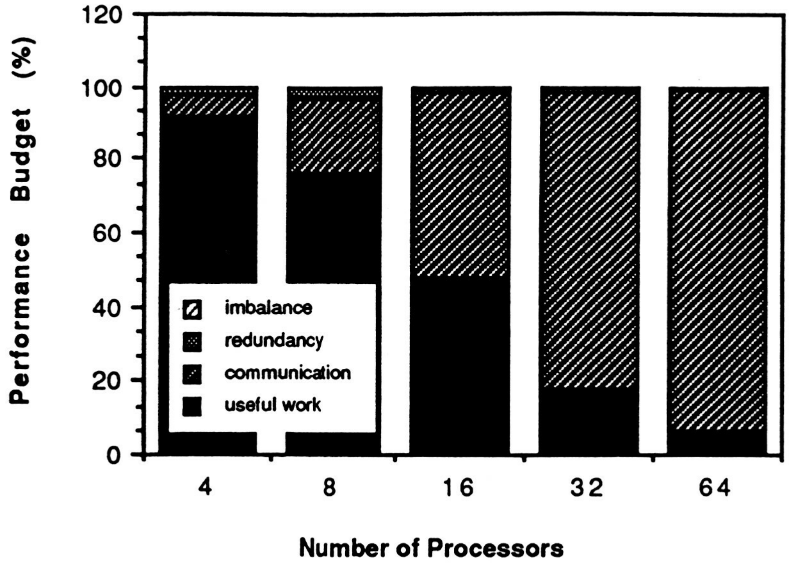


Figure 25. PIC Performance Budget for 2M particles and m=64.

On the T3D, PIC simulation presents a better overall "picture". in terms of execution times. The iteration time is about 30% of that on the Paragon, although the NX routines are considered to be superior to PVM calls. Clearly, processor and network speed are dominant factors. Figures 19 and 20 indicate that the scalability is greatly affected by the communication component, a fact supported by performance budget figures as well. When the ratio of the data to be processed to the number of grid points to be communicated is large enough, the code scales quite well (figure 19). Communication time exhibits a smooth slope with increasing data sizes. The performance budget figures, on the other hand, include smaller portions of useful work than ones on the Paragon, showing the negative effect of PVM. Characteristically, the execution accross the processors are again well-balanced and redundancy is negligibly small.

5. OBSERVATIONS AND CONCLUSIONS

In addition to the presented measurements, our day-to-day experience throughout this study has revealed many issues. We sum the experiences and observations with the following conclusions.

5.1. Effect of Programming Model

The use of the manager-worker model in the N-body code has resulted in many experiences that are quite different from those with worker-worker model used in PIC. In N-body, the model used resulted in some imbalance overhead, although the workload was intentionally balanced using the costzone method. The observed imbalance was due to the focal point of communication created by the manager and the variability of the communication distances from arbitrary nodes to the manager. On the other hand, the worker-worker model balanced the communications in the PIC implementation. However, balanced communication when coupled with many-to-many communications transactions means increased conflicts and performance loss as the case in the original gsum implementation.

5.2. Effect of Memory Management

Again, superlinear scalability can be observed due to improved caching and less frequent paging in parallel system. It would be of interest to investigate a scalability model which takes such memory-related factors into account.

5.3. Effect of Programming Style

Parallel program performance seems to be unusually susceptible to programming style. We have already addressed the effect of using different programming models. In addition, it has been noted during this study that depending on the programmer, some types of overhead can become more dominant than others. In fact, in many cases, reducing one type of overhead comes at the expense of increasing other types of overhead. For example, in many cases communications can be replaced by redundancy and vice versa. A general rule, however, is that redundancy is cheaper than communications, in most cases.

5.4. Physical Effects

One phenomena that was observed in the Paragon, was that the speed of a specific problem might differ based on which partition of the machine is used. This was the case even when the same number of nodes and the topology of the partition is maintained. After repeated measurements and investigations, it was found that processors that are physically closer to the cooling system tend to run slower than those that are farther away [13]. Up to 7% variability in execution time was observed and attributed to this phenomena.

5.5. CONCLUSIONS

In this study some of the sources of overhead were identified and measured for real applications selected from NASA ESS domain. Among the observed sources of overhead are the programming model, programming style, and the communications patterns. With the sophistication of multicomputers and in the light of the lack of comparably powerful compiler technology, parallel machines are much less forgiving than uniprocessor environments. Subtle changes in programs can increase or decrease overhead significantly. Some types of overhead can be reduced by following better programming practices and some can be reduced by converting them to less costly overhead activities. The dominant type of overhead is communications and could be in many cases a real challenge to scalability. While it is not considered a good programming practice, duplication redundancy can effectively help reduce the effect of communications. Efficiency in most of the cases, specially when data sets were large enough, was greater than 50% which indicates that progress in high-performance computing is consistent with the needs of scientific parallel simulations.

References

[1] A. W. Appel, "An Efficient Program for Many-Body Simulation", SIAM J. Sci. Stat. Computing, vol. 6, 1985.

[2] J. Barnes and P. Hut, "A Hierarchical O(N.logN) Force-Calculation Algorithm," Nature, vol. 324, pp. 446-449, 1986.

[3] C. K. Birdsall and A. B. Langton, Plasma Physics via Computer Simulation, McGraw-Hill Inc., New York, 1985.

[4] P. M. Campbell, E. A. Carmona and D. W. Walker, "Hierarchial Domain Decomposition With Unitary Load Balancing for Electromagnetic Particle-In-Cell Codes", Proceedings of the Fifth Distributed Memory Computing Conference, 1990.

[5] R. D. Ferraro, P. C. Liewer and V. K. Decyk, "Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code", Journal of Computational Physics, vol.109, no.329, 1993.

[6] G. Fox, Numerical Algorithms for Modern Parallel Computer Architectures, pp. 37-62, Springer-Verlag, 1988.

[7] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, Prentice Hall, Englewood Cliffs, NJ, 1988.

[8] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations", J. Comp. Phys., vol.73, pp. 325-348, 1987.

[9] F. H. Harlow, "The Particle-in-Cell Computing Method in Fluid Dynamics", Methods Comput. Phys., vol.3, no.319, 1964.

[10] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles. Adam Hilger, 1988.

[11] J. Katzenelson. Computational Structure of the N-Body Problem," SIAM J. Sci. Stat. Comput., vol. 10, no. 4, pp. 787-815, 1989.

[12] P. C. Liewer and V. K. Decyk, "A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes", Journal of Computational Physics,vol.85,no.302, 1989.

[13] Eric Mira, Personal Communication, Intel SSD, December 1994.

[14] Paragon User's Guide (312489-002), Intel Corporation, Beaverton, Oregon, 1993.

[15] J. Salmon, "Parallel N log N N-Body Algorithms and Applications to Astrophysics," COMPCON, Spring 1991.

[16] J. P. Singh, Parallel Hierarchical N-Body Methods and Their Implications, Ph.D. thesis, Stanford University, February 1993.

APPENDIX C:

WORKLOAD CHARACTERIZATION

In Collaboration with A. Meajil (GWU) and T. Sterling (CESDIS)

A Quantitative Approach for Representing and Differentiating Parallel Architectures Workloads

Abdullah Meajil*, Tarek A. El-Ghazawi*, and Thomas Sterling+

*Department of Electrical Engineering and Computer Science
The George Washington University
Washington, D.C. 20052

+Center of Excellence in Space Data and Information Sciences
Goddard Space Flight Center
Code 930.5
Greenbelt, MD 20771

Corresponding Author: Tarek El-Ghazawi, tarek@seas.gwu.edu, (202)994-5507

Abstract

Experimental design of parallel computers calls for quantifiable methods to compare and evaluate the requirements of different workloads within an application domain. Such metrics can help establish the basis for scientific design of parallel computers driven by applications needs, to optimize performance to cost. In this paper, a framework is presented for representing and comparing workloads, based on the way they would exercise parallel machines. This workload characterization is derived from parallel instruction centroid and parallel workload similarity. The centroid is a workload approximation which captures the type and amount of parallel work presented by the workload on the average. The workload similarity is based on measuring the normalized Euclidean distance between workload centroids. It will be shown that this method outperforms comparable ones in accuracy as well as in time and space requirements. An analysis of the NAS parallel benchmark workloads will be presented in order to demonstrate the utility and insight provided within this framework.

Key words: *Instruction-Level Parallelism; Benchmarking; Workload Characterization; Performance Evaluation*

1. Introduction

The design of parallel architectures should be based on the requirements of real-life production workloads, in order to maximize performance to cost. One essential ingredient in this is to develop scientific basis for the design and analysis of parallel benchmark suites. The design of such parallel benchmark suites should be founded on criteria that establish an association between the benchmark test suite and the target production workloads it represents. Selection of test workloads from an application domain must be determined by specific metrics that delineate the salient equivalencies and distinctions among candidate test codes. Previous efforts have addressed the problem of characterizing and measuring specific aspects of parallel workloads. Depending on the irrespective objectives, these efforts have quantified different attributes such as the total number of operations, average degree of parallelism, or instruction mixes [1-15]. However, unresolved still is the means to characterize parallel workloads based on how they are expected to exercise parallel architecture. Such characterization has to be valid across a wide-range of parallel architectures, in order to focus on the applications and their requirements. Therefore we propose an application characterization which takes into account the type of operations and operation counts presented to the machine on a cycle-by-cycle basis, as given by the dynamic parallel instruction sequence in workloads. With this in place, the similarity among each pair of workloads is measured using the normalized Euclidean distance, a computationally efficient technique for pattern matching.

Since measuring parallel instructions is of interest to this study, we consider efforts that examined instruction-level parallelism. Researchers have measured instruction-level parallelism to try different parallel compilation concepts and study their effect on parallelism. Most of these studies measured the limits of (average) parallelism under ideal conditions, such as the oracle model where parallelism is only limited by true flow dependencies. Then, they examined the drop in parallelism when specific architectural or compilation implementation concepts were introduced into the model.

Studies on instruction-level parallelism have taken one of two approaches. One approach is to analyze the selected workload statically at the source-code level (or object-code with a special interpreter based on a certain machine) [1-3]. The other approach is to collect dynamic traces from actual execution and schedule the instructions on the target machine model [4-12]. The static analysis of workloads tends to give conservative estimates for available parallelism since control dependencies can be only resolved at run time. On the other hand, the dynamic analysis of workloads using speculative execution and branch prediction [16] can measure the amount of parallelism which theoretically exists in a given workload. Although the scope was different in these studies, the techniques are of interest to our work as alternative means of measuring parallelism. Many researchers have observed that benchmarking should become more of a scientific activity [17]. Due to the necessity of parallelism for achieving good performance, this work develops a well founded parallelism-based workload representation and comparison framework. This framework provides meaningful information to designers and users of high-performance systems as well as to parallel benchmarking developers and performance analysts.

In this work we only consider workload characterization based on parallel instructions, which encompasses information on parallelism, instruction mix, and amount and type of work on a cycle-by-cycle basis. Bradley and Larson [18] have considered parallel workload characterization using parallel instructions. Their technique compares the differences between workloads based on executed parallel instructions (*EPI*). Executed parallelism is the parallelism exploited as a result of interaction between hardware and software. This technique is, therefore, an architecture-dependent technique due to its dependency on the specific details of the underlying architecture. In their study, a subset of the Perfect' Benchmarks has been chosen to run on the Cray Y-MP. Then a multidimensional matrix that represents the workload parallelism profile was constructed. The Frobenius matrix norm [29] is then used to quantify the difference between the two workload parallelism matrices. In addition to requiring a lot of space and time, this method is restricted to comparing identical executed parallel instructions only.

The technique proposed here, in contrast, uses the vector-space model [19] to provide a single point (but multidimensional) representation of parallel workloads and measure the degree of similarity between them. The similarity is derived from the spatial proximity between workload points in that space and therefore provides a collective meaure of similarity based on all instructions. In specific, each workload in a benchmark suite is approximated by a parallel-instruction centroid under this model. The difference between two workloads are quantified using appropriately normalized Euclidean distance between the two centroids.

Architecture-invariance of our parallel-instruction vector-space model is derived from using the oracle abstract architecture model [4, 12]. In order to simulate such an oracle, two major modules are used: an interpreter and a scheduler [12, 25]. The interpreter accepts the assembly instructions generated from high-level code and executes it. The stream produced is passed to the scheduler which places each instruction at the earliest possible level for execution, based on the dependencies between the current instruction and the previously scheduled ones. The "two pass" nature of this process gives us the oracle. Or loosely speaking is to assume that an *Oracle* is present to guide us at every conditional jump, telling us which way the jump will go each reference, and resolving all ambiguous memory references [4]. Therefore, the oracle model is an idealistic model that considers only true flow dependencies. The parallel instructions (*PI*) are generated by scheduling sequential instructions that are traced from a RISC processor execution onto the oracle model. The traced instructions are packed into parallel instructions while respecting all flow dependencies between instructions. To compare our technique with the parallelism-matrix method, we consider an extended version of the parallelism-matrix technique which is made architecture-invariant by replacing the Cray Y-MP simulator with the oracle model.

In this paper we present the concept of parallel-instruction vector space model and a parallel-instruction workload similarity measurement technique. We compare this technique to the parallelism-matrix method [18]. It will be shown here that our method is machine-invariant and better represents the degree of similarity between workloads. Further, the technique is very cost efficient when compared with similar methods. We also show that the parallel-instruction vector space model provides a useful framework for the design and analysis of benchmarks. This is demonstrated by analyzing some of the NAS Parallel Benchmark workloads [20, 21] and their performance measurements using this model. The NAS Parallel Benchmark (NPB) suite is rooted in the problems of computational fluid dynamics (CFD) and computational aerosciences. It consists of eight benchmark problems each of which is focusing on some important aspect of highly parallel supercomputing, for aerophysics applications [22, 23]. This paper is organized as follows. Section 2 presents an overview of previous work, while section 3 presents our parallel-instruction vector space model in details. The comparison between the two techniques is discussed in section 4. Experimental measurements for the NAS Parallel benchmarks will be presented in section 5. Finally, conclusions are given in section 6.

2. The Parallelism-Matrix Workload Representation

This technique represents an executed-parallelism workload profile in a multidimensional matrix (*n*-matrix). Each dimension in this *n*-matrix represents a different instruction type in a workload. "Work", in this section, has been defined to be the total number of operations of interest a workload can have [18]. When there is only one instruction type of interest, work is considered to be the total operations of that type in a workload. Therefore, a natural extension to the simple post-mortem average is a histogram $W = \langle W_0, \ldots, W_f \rangle$, where W_i is the number of clock periods during which i operations of interest type were completed simultaneously. The sum

$$t = \sum_{i=0}^{f} W_i \qquad (1)$$

is the number of clock periods consumed by the entire workload, and the weighted sum

$$w = \sum_{i=0}^{t} iW_i \qquad (2)$$

is the total amount of work performed by the workload. To facilitate comparisons between workloads that have different execution times, each entry in the histogram is divided by t, the total execution time in clock periods, to produce a normalized histogram called the *parallelism vector* $P = \langle P_0, ^{\varrho}, P_f \rangle$, where $P_i = W_i/t$. By construction, each entry P_i has a value between 0 and 1 that indicates the fraction of time during which i units of work were completed in parallel.

In a similar way, executed parallelism matrices of arbitrary dimension can be constructed with one dimension for each of the different kinds of work that are of interest. Some other possibilities for "work" include logical operations, integer operations, and I/O operations. Depending on how work is defined, various parallelism profiles from an executed-parallelism workload matrix can be obtained.

To illustrate a two-dimensional case, Figure 1 shows a two-dimensional matrix that represents a parallelism profile for ARC2D workload which represents the Aerodynamics application area in the Perfect' Benchmarks suite. This parallelism matrix has been the output of the CRAY Y-MP simulator that has three floating-point functional units (add, multiply, and reciprocal approximation) and three memory units (two load and one store). In this matrix the row index represents the multiplicity of memory operation in a parallel instruction, while columns represent the multiplicity of the floating-point operation. Each cell position, therefore, indicates a possible mix in a parallel instruction. The value in the cell, however, indicates the fraction of such instruction in the workload. Taking cell (3,0), e.g., it indicates that 1% of the parallel instruction contain 3 memory instructions but no floating-point operation. In this example, the vector of row sums of the parallelism matrix gives a profile of the memory executed parallelism. On the other hand, the vector of column sums shows the profile of floating-point executed parallelism.

			FP		
		0	*1*	*2*	*3*
M	*3*	0.01	0.03	0.03	0.00
E	*2*	0.05	0.13	0.08	0.00
M	*1*	0.07	0.20	0.11	0.00
	0	0.07	0.13	0.07	0.02

Figure 1: *Parallelism matrix for ARC2D in the Perfect' Benchmark suite.*

The parallelism profiles for two workloads, thus far, can be compared by comparing the parallelism matrices for each workload using the Frobenius matrix norm [29] to quantify the difference. If A is the two-dimensional $m \times n$ parallelism matrix for workload 1 and B is the $m \times n$ parallelism matrix for workload 2, then the difference in executed parallelism between the two workloads can be gauged by

$$Diff(A,B) = \| A - B \|_F$$

$$= \sqrt{\sum_{i=0}^{m} \sum_{j=0}^{n} | a_{ij} - b_{ij} |^2} \qquad (3)$$

Intuitively, the Frobenius norm represents the "distance" between two matrices, just as the Euclidean formula is used to measure the distance between two points. This distance may range from *0.00*, for two workloads with identical executed parallelism distributions, to $\sqrt{2}$ in the case where each matrix has only one non-zero element (with value *1.00*) in a different location. Thus, the numbers produced by this method do not scale in way that can provide an intuitional understanding to the degree of similarity. Further, should two workloads be 100% dissimilar, they can still produce different numbers. Additional problems that relate to the processing and memory cost of this method are addressed in section 4.

3. The Parallel-Instruction Vector-Space Model

Our Parallel-Instruction Vector Space Model is presented here provides for an effective workload representation (Characterization), as will be shown. Effectiveness, in this regard, refers to the fidelity of the representation and the associated space and time costs. In this framework, each parallel instruction can be represented by a vector in a multidimensional space, where each coordinate corresponds to a different instruction/operation type (***I-type***) or a different basic operation (ADD, LOAD, FMUL, ...). The position of each parallel instruction in the space is determined by the magnitude of the *I-types* in that vector.

Parallel Workload Instant and Parallel Work: The workload instant for a parallel computer system is defined here as the types and multiplicity of operations presented for execution by an idealistic system (oracle model), in one cycle. A workload instant is, therefore, represented as a vector quantity (parallel instruction) where each dimension represents an operation type and the associated magnitude represents the multiplicity of that operation in the parallel instruction. Parallel workload of an application is the sequence of instances (parallel instructions) generated from that application.

Workload Centroid: The centroid is a parallel instruction in which each component corresponds to the average occurrence of the corresponding operation type over all parallel instructions in the workload. Centroid, therefore, can be thought of as the point mass for the parallel workload body.

Workload Similarity: Two workloads exhibited by two applications are, thus, considered identical if they present the machine with the same sequence of parallel instructions. In this case both workloads are said to be exercising the machine resources in the same fashion.

3.1 The Vector-Space Workload Representation Model

Consider three types of operation (*I-types*) such as arithmetic operations (INT), floating-point operations (FP), and memory access operations (MEM), then the parallel-instruction vector can be represented in a three-dimensional space as a triplet:

$$PI = (MEM, FP, INT).$$

If a parallel instruction in a workload is given by

$$PI_i = (4, 7, 2),$$

then this ith parallel instruction in the workload has 4 MEM operations, 7 FP operations, and 2 INT operations. The total operations in this parallel instruction would be 13 operations that can be run simultaneously. In general, parallel instructions are represented as t-vectors of the form

$$PI_i = (a_{i1}, a_{i2}, \dots, a_{it}) \qquad (4)$$

where the coefficient a_{ik} represents the count of operations of type k in parallel instruction PI_i.

Comparing workloads based on sequence of parallel instructions could be quite complex and prohibitive, for realistic workloads. This is because the comparison requires examining each parallel instruction from one workload against all parallel instructions in the other workload, which has very high computational and storage requirements. This has led us to propose the concept of centroid for workload representation and comparison, which is a cost-effective means to represent workloads, see Figure 2.

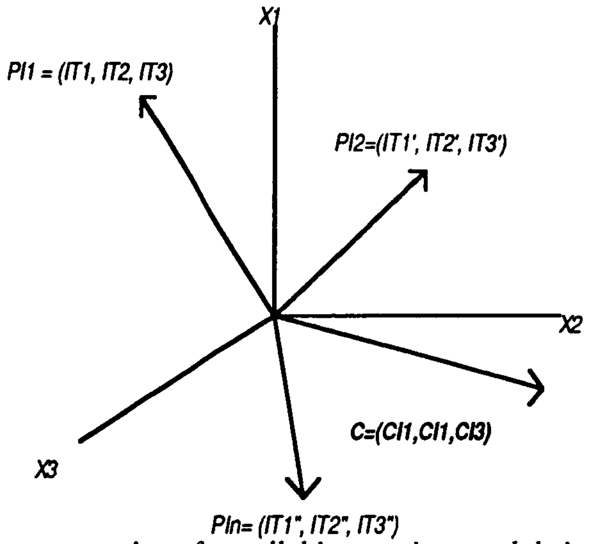


FIGURE 2: *Vector representation of parallel instructions and their centroid in a 3-space.*

The centroid is a parallel instruction in which each component corresponds to the average occurrence of the corresponding operation type over all parallel instructions in the workload. Given a set of *n* parallel instructions constituting a certain workload, the corresponding centroid vector is

$$C = (CI_1, CI_2, \dots, CI_t) \qquad (5)$$

where: $$CI_k = 1/n \sum_{i=1}^{n} a_k \qquad (6)$$

To illustrate the above, Figure 3 shows the steps to generate a centroid vector for a workload.

I(1), I(2), I(3), I(4), I(5), ... , I(22), I(23), I(24)

(a): *Collect dynamic trace from a RISC machine.*

	I-Type$_1$	*I-Type$_2$*	*I-Type$_3$*	*I-Type$_4$*	*I-Type$_5$*
PI$_1$	1	3	0	4	0
PI$_2$	0	2	0	3	1
PI$_3$	0	7	0	2	1

(b): *Schedule on oracle to generate stream of parallel instructions.*

	I-Type$_1$	*I-Type$_2$*	*I-Type$_3$*	*I-Type$_4$*	*I-Type$_5$*
C	1/3	4	0	3	2/3

(c): *Obtain centroid for PIs in a workload.*

Figure 3: *Example of the workload representation.*

In addition to simplifying the analysis, centroids have the quality of providing an easy way to grasp the workload characteristics and the corresponding resource requirements. This is because the centroid couples instruction-level parallelism and instruction mix information to represent the types and multiplicity of operations that the machine is required to perform, on the average, in one cycle. This also represents the functional units types and average number of them needed in the target machine in order to sustain a performance rate close to the machine's peak rate, under such kind of workloads. Due to their simplicity and physical significance, as discussed above, centroids are used in the rest of this work as the basis for workloads representation and comparisons.

3.3 Workload Comparison Using the Vector-Space Model

Measuring similarity based on centroids mandates the selection of a similarity metric which can generate easy to understand numbers. To do so, we propose the following metric characteristics:

(1) Metric generates normalized values between 0 and 1.

(2) "0" represents one extreme (e.g. similar), while "1" represents the other extreme (e.g. dissimilar).

(3) Scales appropriately between these two extremes as the similarity between the compared workloads changes.

This leads us to select the normalized Euclidean distance between two centroids, representing two different workloads, as follow. Let point u be the t-tuple $(a_1, a_2, ^{\circ}, a_t)$ and point v be the t-tuple $(b_1, b_2, ..., b_t)$; then the Euclidean distance, from Pythagoras' theorem, is

$$d(u,v) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a - b_t)^2} \qquad (7)$$

In order to conform with the aforementioned metric characteristics, the distance between any two workloads in a benchmark suite can be normalized by dividing the distance between the two workloads by the maximum distance found in that two workloads, from the origin. Let WL_r and WL_s be two workloads in a benchmark suite, where each can be characterized by a t-centroid vector (t instruction types) as follows:

$WL_r = (CI_{r1}, CI_{r2}, ..., CI_{rt})$, and

$WL_s = (CI_{s1}, CI_{s2}, ..., CI_{st})$.

And let CI_{ik} represent the centroid magnitude of the kth instruction type in workload i. The maximum centroid-vector in this workloads can be represented as follows.

$$C_{max}(WL_r, WL_s) = (max(CI_{r1}, CI_{s1}), \ldots, max(CI_{rt}, CI_{st})) \quad (8)$$

Then, the similarity between the two workloads can be measured as:

$$Sim(WL_r, WL_s) = d(WL_r, WL_s) / d(C_{max}, null\text{-}vector) \qquad (9)$$

where null-vector (origin) is a t-vector in which each element equals to 0; hence, *null-vector* $= (0,0,...,0)$. In this case, 0 represents identical workloads while 1 represents orthogonal workloads that use different operations and thus, would exercise different aspects of the target machine.

4. *Comparison Study for the Two Techniques*

4.1 Examples

Sample examples have been developed in order to demonstrate how the two techniques compare. Let us have a benchmark suite that consists of five workloads $(WL_1, \ldots, WL_6)$. Each workload is presented in a table of size $i \times j$, where i is the total number of unique parallel instructions in the workload and j has a length of $t + 1$. Each one of the t columns represents an operation type (MEM, FP, INT). The additional column, *#PIs*, represents the total number of instances for that unique parallel instruction. For example, *#PIs* = 5 means that there are five instances of a particular parallel instruction in a workload.

<u>*Workload-1:*</u>

#PIs	*MEM*	*FP*	*INT*
5	1	0	1
3	0	1	0
7	1	0	0
2	0	0	1

<u>*Workload-2:*</u>

#PIs	*MEM*	*FP*	*INT*
2	0	1	1
3	1	1	0
7	1	0	1
5	1	1	1

<u>*Workload-3:*</u>

#PIs	*MEM*	*FP*	*INT*
5	3	2	1
7	4	3	0

<u>*Workload-4:*</u>

#PIs	*MEM*	*FP*	*INT*
3	4	3	2
7	3	4	2

2	2	3	1	2	4	4	1
3	2	3	0	5	4	4	2

Workload-5:

#PIs	MEM	FP	INT
3	0	2	0
7	2	0	0
5	1	0	2
2	0	0	2

4.2 Parallelism-Matrix Measurements

In the parallelism-matrix technique, each workload parallelism profile is presented in a three-dimensional matrix (since there are only three types of operations). For example, workload WL_3 is illustrated in Figure 4 by spreading the *INT*-dimension layers of the three-dimensional matrix over two layers for readability.

		FP			
		0	*1*	*2*	*3*
	4	0.00	0.00	0.00	0.412
M	*3*	0.00	0.00	0.00	0.00
E	*2*	0.00	0.00	0.00	0.176
M	*1*	0.00	0.00	0.00	0.00
	0	0.00	0.00	0.00	0.00

(a): *Parallelism matrix when 0 INT-type used.*

		FP			
		0	*1*	*2*	*3*
	4	0.00	0.00	0.00	0.00
M	*3*	0.00	0.00	0.294	0.00
E	*2*	0.00	0.00	0.00	0.118
M	*1*	0.00	0.00	0.00	0.00
	0	0.00	0.00	0.00	0.00

(b): *Parallelism matrix when 1 INT-type used.*

Figure 4: *Parallelism-matrix representation for workload WL_3.*

Figure 4.a represents the *1*st INT layer where no INT operations are in the corresponding parallel instruction. Figure 4.b represents the *2*nd INT layer when only one INT operation is in the parallel instruction.

To compare two workloads, the Frobenius matrix norm [29] is used in order to quantify the distances or differences. Recall that the parallelism-matrix technique has been extended to be architecture-invariant for comparisons with the parallel-instruction vector space model. As mentioned before, the Frobenius norm ranges between *0.00* and $\sqrt{2}$, therefore, it will be divided by that value. Table 1 presents similarity measurements for some pairs of workloads in the benchmark set.

Table 1: *Similarity measurements using parallelism-matrix technique.*

	Parallelism-Matrix

$WL_1 \& WL_2$	0.424
$WL_1 \& WL_3$	0.549
$WL_1 \& WL_4$	0.549
$WL_1 \& WL_5$	0.549
$WL_3 \& WL_4$	0.549

4.3 Parallel-Instruction Vector Space Measurements

In the aforementioned vector-space model, each workload centroid is calculated from all the parallel instructions that are in a workload. All workload centroids are presented in Table 2 where the row represents different workload and the column represents the instruction type in the workload.

Table 2: *Workload centroids.*

	MEM	*FP*	*INT*
WL_1	3.037	*5.626*	*8.068*
WL_2	*7.980*	*17.129*	*13.483*
WL_3	3.12	2.71	0.412
WL_4	*31.115*	*8.885*	*58.462*
WL_5	80.825	53.664	57.212
WL_6	*22.091*	*33.622*	*30.039*

Recall that the distance between two workload centroid points needs to be normalized to produce numbers in the range of *0.00* and *1.00*. The following is the maximum centroid-vector found in the two benchmarks, $WL_2$and WL_3.

Maximum centroid-vector :

$$C_{max}(WL_2, WL_3) \;=\; (3.12, 2.71, 0.824)$$

Therefore, the maximum distance is:

$$d(C_{max}(WL_2,WL_3),null\text{-}vector) = \sqrt{(3.12-0)^2+(2.71-0)^2+(0.824-0)^2}$$
$$= 4.214$$

To compare the two workloads by the normalized Euclidean distance function is employed as follows.

$$d(WL_2,WL_3) \;=\; \sqrt{(3.12-0.883)^2+(2.71-0.589)^2+(0.412-0.824)^2}$$
$$= \sqrt{9.67255\text{‹}} = 3.110073$$
$$Sim(WL_2,WL_3) = 3.110073 \,/\, 4.214 \; = 0.738$$

Table 3 presents similarity measurements for some pairs of workloads in our benchmark suite when the parallel-instruction vector space model is used. Note that *1.00* means dissimilar and *0.00* means identical.

Table 3: *Similarity measurements using parallel-instruction vector space technique.*

	Parallel-Instruction Vector Space
$WL_1 \& WL_2$	0.45318
$WL_1 \& WL_3$	0.8425
$WL_1 \& WL_4$	0.8751
$WL_1 \& WL_5$	0.1804
$WL_3 \& WL_4$	0.65

4.4 Discussion

The similarity among the workloads in the example suite can be examined quantitatively using similarity functions, expressions (3) and (9). Table 4 shows the quantitative similarity for some pairs of workloads when the two techniques are used. Note that similarity in parallelism is not a transitive relation.

Table 4: *Workload similarity in the example-benchmarks with the two techniques.*

	Parallelism-Matrix	*Parallel-Instruction Vector Space*
$WL_1 \& WL_2$	0.424	0.45318
$WL_1 \& WL_3$	0.549	0.8425
$WL_1 \& WL_4$	0.549	0.8751
$WL_1 \& WL_5$	0.549	0.1804
$WL_3 \& WL_4$	0.549	0.65

Evidently, measurements obtained by parallelism-matrix technique have more shortcomings. For example, in the parallelism-matrix the similarity value of the workload comparisons WL_1 & WL_3, WL_1 & WL_4, and WL_1 & WL_5 are all 0.549. This value does not change in these cases because of the absence of the identical parallel instructions from the workloads. However, if there are some identical parallel instructions in workloads, then the similarity value may have more meaningful values. For example, in comparing WL_1 & WL_2 the similarity value is 0.424. The reason is that, both workloads have a common identical parallel instruction. Hence, the parallelism-matrix technique lacks the ability to compare realistic workloads when they lack identical parallel instructions. This is the case even when the parallel instructions of the two workloads are quite similar but not identical. In the parallel-instruction vector space technique, Table 4 shows more meaningful values. When two workloads are quite different, the similarity values are high as in the case of WL_1 & WL_4. On the other hand, when there are some differences in the workloads, the similarity value changes proportionally. For example, WL_1 and WL_2 behave almost in the same manner. By applying the parallel-instruction vector space technique the similarity between these two workloads equals 0.45318, while 0.549 is produced by the parallelism-matrix technique. A similar scenario occurs when WL_1 & WL_5 are compared.

In general, the parallel-instruction vector space method presents more detailed information. For each workload centroid, each attribute represents an arithmetic mean of a type of instruction in the workload. By comparing this centroid to other workload centroid, each matching attribute will be compared. This comparison tells in which direction these two workloads are different. Considering workloads WL_1 and WL_3, along the arithmetic instruction type, these two workloads exercise the oracle model in the same manner. However, at the floating-point instruction type, WL_3 uses more floating-point functional units than WL_1.

The parallel-instruction vector space method is also more efficient in time and space. After producing parallel instructions, both techniques make two steps in order to measure the workload similarity. The first step is workload representation, and the second is workload comparison. The parallelism-matrix technique represents a workload in a t-dimensional matrix where each dimension represents an instruction type. The maximum magnitude of a dimension is $n + 1$, where n represents the maximum instruction type occurrences in any parallel instruction in that workload. Therefore, the parallel matrix technique needs as much storage as the size of the matrix. This has storage complexity of $O(n^t)$. On the other hand, the parallel-instruction vector space model represents a workload by a centroid of length t. Therefore, the storage complexity of this technique is $O(t)$. The time for workload representation, in the parallelism-matrix technique, takes the parallel-instruction counts (p) times the parallel-instruction length (t), or $O(p \Diamond t)$. This is because all parallel instructions have to be generated first, before constructing and filling the matrix. However, in the parallel-instruction vector space model, the computational complexity is $O(t)$. This is due to the fact that the workload centroid is calculated on-the-fly.

In the comparison step (measuring similarity), the parallelism-matrix technique compares every element of one matrix with the corresponding element in the other matrix. Therefore, the computational complexity of this technique is $O(n^t)$. In the parallel-instruction vector space model, however, the computational complexity is $O(t)$. This is due to the fact that the workload centroid has t types of instructions.

Table 5 summarizes the comparative study between the parallelism-matrix technique and the parallel-instruction vector space technique. It shows that our parallel-instruction vector space model outperforms the parallelism-matrix technique for measuring the workload similarity in all essential aspects.

Table 5: *Comparison between Parallelism-Matrix and Vector-Space Model.*

	Parallelism-Matrix	***Parallel-InstructionVector Space***
Representation Cost (time)	$O(p.t)$	$O(t)$
Representation Cost (storage)	$O(n^t)$	$O(t)$
Comparison-Cost	$O(n^t)$	$O(t)$
Accuracy	Depends on identical *PIs*	Depends on all *PIs*
Machine-Dependency	Architecture-dependent†	Architecture-invariant

† Original parallelism-matrix technique [18].
p: parallel-instruction count.
t: parallel-instruction size.
n: maximum dimension length.

5. NAS Parallel Benchmark Experimental Workload Comparison

In order to demonstrate the potential utility of this model and verify the underlying concepts with real-life applications we consider to study the NAS Parallel Benchmark suite [22, 23] using our model. We start by representing the workloads in this suite using parallel instruction centroids. Then we characterize the similarity among different workload pairs using the normalized Euclidean distance approach. Finally, we demonstrate that in real-life workloads, such as those of NPB, parallelism smoothability is high enough to use average degree of parallelism to represent parallel activities, as in the centroid.

5.1 A NAS Parallel Benchmark Overview

The NPB suite consists of two major components: five parallel kernel benchmarks and three simulated computational fluid dynamics (CFD) application benchmarks. This benchmark suite successfully addresses many of the problems associated with benchmarking parallel machines. They intended to accurately represent the principal computational and data movement requirements of modern CFD applications. An exhaustive description of these NPB problems is given in [20-23].

In order to keep traces and analysis time within practical limits, we have used the short input files provided by the NAS Parallel Benchmark suite. The sample codes, provided by NAS, actually solve scaled-down versions of the benchmarks that run on many current-generation workstations. The standard input sizes for the NPB suites referred to as the Class A and Class B size problems. Table 6 lists the problem size [20] and the dynamic operation counts of: the sample code problems, the Class A problems, and Class B problems. Operation counts are obtained using the *spy* tool [24].

Table 6: *Operation Counts for NAS "Sample, Class A, and Class B" Benchmarks running on One Processor.*

Benchmarks	***Problem Size***			***Dynamic Operation Count (10^9)***		
	Sample	***Class A***	***Class B***	***Sample***	***Class A***	***Class B***
embar	2^{24}	2^{28}	2^{30}	8.9811	26.68	1008.8
mgrid	32^3	256^3	256^3 *U*	0.1154	3.905	18.81
cgm	10^5	14,000	75,000	0.5103	1.508	54.89
fftpde	64^3	256^2 x128	256^2 x 512	1.5230	5.631	71.37
buk	2^{16}	2^{23} x 2^{19}	2^{25} x 2^{21}	0.0768	0.7812	3.150
applu	12^3	64^3	102^3	0.5200	64.57	319.6
appsp	12^3	64^3	102^3	0.8920	102.0	447.1
appbt	12^3	64^3	102^3	1.1157	181.3	721.5

U Code is different from Class A [20].

5.2 Experimental Measurements Tool and Process

In order to explore the inherent parallelism in workloads, instructions traced are scheduled for the oracle model architecture. This model presents the most ideal machine that have unlimited processors and memory, and does not incur any overhead. The Sequential Instruction Trace Analyzer (SITA) is a tool developed at McGill University by Kevin Theobald to measure the amount of parallelism which theoretically exists in a given workload [12, 25]. SITA takes a dynamic trace generated by spy tool from a sequential execution of a conventional program, and schedules the instructions according to how they could be executed on an idealized architecture while respecting all relevant dependencies between instructions. Currently, SITA is used to analyze SPARC executables and is designed to work with spy tool, which is the only tool needed from the Spa package [24]. SITA tool includes a pre-analyzer (*sitapa*), a control-dependence analyzer (*sitadep*), and a trace scheduler(*sitarun*). Note that traced processes have been observed to run about 40 times slower than normal. If spy is used with a trace analyzer, such as sitapa or sitarun, the resulting system will run some 400-600 times slower than normal (400 for oracle and 600 for other models).

The analysis process of a SPARC workload or benchmark takes four steps. First, a SPARC executable file is created, using the desired optimization level. The results will be more meaningful if the program is statically linked. This eliminates the spurious instructions used in linking a program to the libraries. Secondly, a pre-analyzer (sitapa) is run with spy and executable to extract a list of basic blocks and frequencies of the workload, which is then read by the control-dependence analyzer (sitadep) to produce an annotated list, as the third step. This annotations include control-dependency relationships between the blocks and destination frequencies. Finally, the scheduler (sitarun) is run with the annotated list as input, and generally with spy and executable. The scheduler produces output indicating the parallelism available for the given input trace under the given oracle model. There are 69 basic instruction operations in SPARC. These instructions mainly fall into five basic categories: load/store (*Memops*), arithmetic/logic/shift (*Intops*), control transfer (*Branchops*), read/write control register (*Controlops*), and floating-point operate (*FPops*). Therefore, each parallel instruction presented by a vector of length five [26].

5.3. Workload Centroids for NAS

Parallel-instruction centroid vectors can reveal differences in workload behavior that can not be distinguished by averages of parallelism degrees as shown in Table 7. Therefore, the parallelism behavior of two workloads can be efficiently compared by using the aforementioned parallel-instruction vector space model and the similarity function, expression (9), to quantify the similarity between these workloads.

Table 7: *Centroid values for the NAS Parallel Benchmarks.*

Benchmarks	***Intops***	***Memops***	***FPops***	***Controlops***	***Branchops***
embar	81.344	59.469	14.369	0.000009	37.337
mgrid	33.857	19.516	0.7958	0.04973	9.22
cgm	4.475	3.798	0.84	0.000012	0.8463
fftpde	184.422	128.224	33.466	10.8513	57.765
buk	2.428	1.735	0.4502	0.000001	0.662
applu	1,031.789	559.136	69.79	0.04813	413.972
appsp	8,260.854	5,262.65	604.75	26.195	3,504.31
appbt	2,788.824	847.519	49.73	4.307	1,065.396

5.4 Similarity Measurements

Table 8 quantifies the similarity between each pair of benchmarks in the NAS Parallel Benchmark suite, using expression (9). Again, note that the similarity in parallelism is not a transitive relation. We first compare appsp and appbt, two workloads that are representative of computations associated with the implicit operators of CFD codes such as ARC3D at NASA Ames. The relatively high value, 0.64, of the similarity in parallelism illustrates that these two workloads have different parallelism behaviors. Next we consider buk, a workload representing the application area of integer sorting, and cgm, this workload is typical of unstructured grid computations.

Table 8: *Similarity values for the NAS Parallel Benchmarks.*

	embar	***mgrid***	***cgm***	***fftpde***	***buk***	***applu***	***appsp***	***appbt***
embar	0.000							
mgrid	0.53	0.000						
cgm	0.943	0.834	0.000					
fftpde	0.39	0.803	0.974	0.000				
buk	0.971	0.918	0.319	0.987	0.000			
applu	0.9066	0.967	0.9954	0.782	0.99756	0.000		
appsp	0.9895	0.99616		0.9772	0.999707	0.8666	0.000	
appbt	0.966	0.987	0.998	0.924	0.999	0.4864	0.64	0.000

The relatively low value of the similarity in parallelism behavior, 0.319, illustrates that these two workloads have relatively similar parallelism properties. Although the two workloads come from different application areas, each workload is expected to exercises target machines with a very similar mix of parallelism. The same conclusion of might be also drawn from the measurement, 0.39, of the similarity in parallelism between embar and fftpde workloads.

5.5 Parallelism Smoothability of NAS and Implications

Smoothability [12] is metric designed to capture the effect of parallelism profile variability around the average degree of parallelism. It is basically the ratio of execution time with no restriction on the number of processors to the execution time when the number of available processors is limited to the average degree of parallelism. Fortunately, SITA has the ability to limit the number of operations which can be packed into one parallel instruction, and thus allow measuring smoothability.

The interest in smoothability stems from the fact that the centroid is based upon the average degree of parallelism for each type of operation. Therefore, for centroids to be a good representation of workloads, those workloads should have relatively high smoothability (close to 1). In this section we show that the majority of real-life applications, such as those represented by NAS, have high smoothability.

In Table 9 we list the parallelism results for the NAS parallel benchmark workloads running on the oracle model including the smoothability values. Our results indicate that the parallelism obtained has a relatively smooth temporal profile which exhibits a high degree of uniformity in the parallelism except for the *buk* benchmark. In all cases, but the *buk* benchmark, the smoothability is better than 70%. In addition, we list the average operation delay, i.e., the average number of parallel instructions by which each operation is delayed before it can be executed. (The average includes instructions which are executed as soon as they are ready, which are counted as 0.) Low numbers, as with *mgrid* and *appbt*, indicate that the application is already fairly evenly distributed in time, so not much "smoothing" is needed. High numbers, as with *embar* and *buk* suggest that many operations are being delayed a long time before being executed. The high smoothability numbers, however, suggest that delays do not significantly increase the lengths of critical paths. Most importantly, in the context of this study, the smooth temporal behavior supports the fidelity of representing practical workloads using parallel instruction centroids.

Table 9: *Smoothability and the effect of finite processors.*

Benchmar	***Smoothabili***	***CPL(•)***	***$P_{Avg.}$***	***CPL(P_{Avg})***	***Avg Op Del***
embar	0.82765	46,650,326	181.867	56,364,246	3498.45
mgrid	0.96794	1,818,865	63.099	1,879,119	268.136
cgm	0.68046	2,697,227	188.654	3,963,866	473.631
buk	0.95090	14,553,108	4.99	15,302,561	1684.723
appbt	0.98638	237,224	4,699.796	240,499	327.973
appsp	0.91282	50,512	17,305.43	55,336	809.516

6. Conclusions

This paper introduced a parallelism-based methodology for an easy to understand representation of workloads. The method is architecture-invariant and can be used effectively for the comparison of workloads and assessing resource requirements. A method for comparing workloads based on the notion of centroid of parallel instruction was introduced. This method uses the normalized Euclidean distance to provide an efficient means of comparing the workloads. The notion of centroid coupled with distance (similarity) among pairs of workloads provide the basis for quantifiable analysis of workloads to make informed decisions on the composition of parallel benchmark suites. Analysis of existing benchmarks is also provided for by this model in which the centroid sheds light on the hardware resource requirements and how the benchmark is exercising the target machines, and the distance among workload pairs allows identifying possible correlation.

A comparative study between the parallelism-matrix technique and our parallel-instruction vector space model was also presented. It was shown that the parallelism-matrix technique depends only on identical rather than similar parallel instructions. However, the introduced parallel-instruction vector space model takes all parallel instructions into account when representing workloads and their similarities. Furthermore, while the parallelism-matrix technique requires *O(p◊t)* computational time for workload representation, the parallel-instruction vector space model requires only $O(t)$. Considering the storage requirements, the parallelism-matrix technique needs $O(n^t)$ memory space, whereas the parallel-instruction vector space model needs only $O(t)$. In addition, when two workloads are compared, the computational cost in the parallelism-matrix technique is $O(n^t)$. On the other hand, the parallel-instruction vector space only requires $O(t)$ computational time. Hence, the parallel-instruction vector space model does not only provide more accurate, but also more cost-effective parallelism-based representation of workloads.

The parallel-instruction workload model was used to study the similarities among the NAS Parallel Benchmark workloads in a quantitative manner. The results confirm that workloads in NPB represent a wide range of non-redundant applications with different characteristics. It was also shown from the NPB results that parallel instruction centroids provide good approximation of workloads due to the fact that most practical workloads have smooth parallelism profiles.

References

[1] D.J. Kuck, Y. Muraoka, and S.C. Chen. "*Measurements of parallelism in ordinary FORTRAN programs*," Computer, vol. 7, pp. 37-46, 1974.

[2] E. M. Riseman and C. C. Foster, "*The inhabitation of potential parallelism by conditional jumps*," IEEE Transactions on Computers, vol. C-21, no. 12, pp. 1405-1411, December 1972.

[3] G.S. Tjaden and M.J. Flynn, "*Detection and parallel execution of independent instructions*," IEEE Transactions on Computers, vol. C-19, no. 10, pp. 889-895, October 1970.

[4] A. Nicolau and J.A. Fisher, "*Measuring the parallelism available for very long instruction word architectures*," IEEE Transactions on Computers, vol. 33, no. 11, pp. 968-976, Nov. 1984.

[5] M. Butler, T Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "*Single Instruction Stream Parallelism Is Greater Than Two,*" in Proceedings of the 8*th* Annual Symposium on Computer Architecture, pp. 276-286, May 1991.

[6] M.S. Lam and R.P. Wilson, "*Limits of control flow on parallelism*," in Proceedings of the 19*th* Annual International Symposium on Computer Architecture, Gold Coast, Australia, pp. 46-57, May 19-21, 1992.

[7] M. Kumar, "*Measuring parallelism in computation-intensive scientific/ engineering applications*," IEEE Transactions on Computers, vol. C-37, no. 9, pp. 1088-1098, Sep. 1988.

[8] D.W. Wall, "*Limits of instruction-level parallelism*," Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 176-188, Santa Clara, California, April 8-11, 1991.

[9] T. Austin and G. Sohi, "*Dynamic dependence analysis of ordinary programs*," Proceedings of the 19*th* Annual International Symposium on Computer Architectures, pp. 342-351, 1992.

[10] Arvind, D.E. Culler, and G.K. Maa, "*Assessing the benefits of Fine-grained parallelism in dataflow programs*," Proceedings of Supercomputing 88, pp. 60-69, November 1988.

[11] A.K. Uht, "*Extraction of massive instruction level parallelism*," ACM SIGARCH News, pp. 5-12, June 3, 1993.

[12] K.B. Theobald, G.R. Gao, and L.J. Hendren, "*On the limits of program parallelism and its smoothability*," Proceedings of the 25th Annual International Symposium on Micro-architecture (MICRO-25), pp. 10-19, Portland, Oregon, December 1992. Also ACAPS Technical Memo 40, McGill University, June 26, 1992.

[13] T. Conte and W. Hwu, "*Benchmark Characterization*," IEEE Computer, pp. 48-56, January 1991.
[14] M. Calzarossa and G. Serazzi, "*Workload Characterization for Supercomputer*," Performance Evaluation of Supercomputers, J.L. Martin (editor), pp. 283-315, North-Holland, 1988.

[15] J. Martin, "*Performance Evaluation of Supercomputers and Their Applications*," Parallel Systems and Computation, G. Paul and G. Almasi (Editors), pp. 221-235, North-Holland, 1988.

[16] J.K.F. Lee and A.J. Smith, "*Branch prediction strategies and branch target buffer design*," Computer, vol. 17, no. 1, pp. 6-22, January 1984.

[17] R. Hockney, *The science of benchmarking*, tutorial handouts, Supercomputer 94, Washington, DC, November 1994.

[18] D. Bradley and J. Larson, "*A parallelism-based analytic approach to performance evaluation using application programs*," Proceedings of the IEEE, vol., 81, no. 8, pp. 1126-1135, August 1993.

[19] E.W. Swokowski, *Calculus with analytic geometry. 3*rd ed., Prindle, Weber & Schmidtg Publishers, Boston, Mass., 1983.

[20] D. Bailey, E. Barszcz, L. Dagum, and H. Simon, "*NAS Parallel Benchmark Results 10-94*," NAS technical report NAS-94-001, October 1994, Calif.: NASA Ames Research Center.

[21] D. Bailey, *The science of benchmarking*, tutorial handouts, Supercomputer 94, Washington, DC, November 1994.

[22] D. Bailey, et al., "*The NAS Parallel Benchmarks*," Int'l J. Supercomputer Applications, Vol. 5, No. 3, Fall 1991, pp. 63-73.

[23] D. Bailey, E. Barszcz, L. Dagum, and H. Simon, "*NAS Parallel Benchmark Results*," IEEE Parallel & Distributed Technology, February 1993, pp. 43-51.

[24] G. Irlam, The Spa package, version 1.0, October 1991.

[25] K.B. Theobald, G.R. Gao, and L.J. Hendren, "*Speculative Execution and Branch Prediction on Parallel Machines*," ACAPS Technical Memo 57, McGill University, December 21, 1992.

[26] The SPARC Architecture Manual, Version 8, SPARC Int'l, Inc., Menlo Park, CA, 1991.

[27] P. Fleming and J. Wallace, "*How Not to Lie With Statistics: The Correct Way to Summarize Benchmark Results*," Communications of ACM, pp. 218-221, March 1986, Vol. 29, No. 3.

[28] G. Amdahl, "*Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities*," in AFIPS Conference Proceedings, 1967, pp. 483-485.

[29] R. Horn and C. Johnson, *Matrix Analysis*. Chapter 5: Norms for vectors and matrices, Cambridge University Press, New York, 1985.

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Experimental Evaluation and Workload Characterization for High-Performance Computer Architectures		5. Report Date 6. Performing Organization Code
7. Author(s) Tarek A. El-Ghazawi		8. Performing Organization Report No.
9. Performing Organization Name and Address Dept. of Elec. Eng. and Computer Science The George Washington University Washington, DC 20052		10. Work Unit No. 11. Contract or Grant No. NAS5-32337 USRA subcontract No. 5555-34
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001 NASA Goddard Space Flight Center Greenbelt, MD 20771		13. Type of Report and Period Covered Final January 1, 1994 - May 31, 1995 14. Sponsoring Agency Code

15. Supplementary Notes

This work was performed under a subcontract issued by
Universities Space Research Association
10227 Wincopin Circle, Suite 212
Columbia, MD 21044 Task 31

16. Abstract

This research is conducted in the context of theJoint NSF/NASA Initiative on Evaluation (JNNIE). JNNIE is an inter-agency research program that goes beyond typica benchmarking to provide in-depth evaluations and understanding of the factors that limit the scalability of high-performance computing systems. Many NSF and NASA centers have participated in the effort. Our research effor was an integral part of implementing JNNIE in the NASA ESS grand challenge applications context. Our research work under this program was composed of three distinct, but related activities. They include the evaluation of NASA ESS high-performance computing testbeds using the wavelet decomposition applicaton; and developing an experimental model for workload characterization for understanding workoad requirements.

In this report, we provide a summary of findings that covers all three parts, a list of the publications that resulted from this effot, and three appendices with the detatils of each of the studies using a key publication developed under the respective work.

17. Key Words (Suggested by Author(s)) Paralllel Processing Experimental Performance Image Processing Wavelets		18. Distribution Statement Unclassified--Unlimited	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 62	22. Price

NASA Form 1626 Oct 86