JKAU: Sci., **vol. 17,** pp. 207-222 (2005 A.D. / 1425 A.H.)

Evaluating Budget of Overhead and Scalability on High-Performance Computing Systems

ABDULLAH I. ALMOJEL
Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
almojel@mohe.gov.sa

ABSTRACT. High-Performance Computing Systems (HPCS) based on parallel processing have the potential for satisfying the rapid growth in computational requirements 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 one of the NASA Earth and Space Sciences (ESS) applications running on the Jet Propulsion Laboratory/Earth and Space Sciences High-Performance Computing (HPC) systems.

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 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[12]. The Joint NSF/NASA Initiative on Evaluation (JNNIE) is a national evaluation activity that involves NSF and NASA centers and, thus, includes a large number of testbeds and applications characterization, usability, macro performance, and micro performance[12]. 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 inter-processor communications overhead, redundancy overhead, load-imbalance overhead, and time spent doing useful work. This set of measurable quantities are referred to as the *performance budget.* The target application was selected from NASA Earth and Space Science (ESS) domain. The measurements presented here are from the N-body simulations. The target high-performance computing platforms for this study were the ESS Cray T3D and Intel Paragon at the NASA Jet Propulsion Laboratory (JPL)[5,8,11].

This paper is organized as follows. Section 2 discusses the selected NASA ESS application as well as the target high performance computing platforms for this study. Section 3 discusses the type of overhead measurements collected under our *Performance Budget model*. Sections 4 and 5 present the measurements collected for the N-body code. Observations are derived in section 6, while conclusions are in section 7.

Application and System Scope

As mentioned earlier, the NASA Earth and Space Science (ESS) application considered in this work is the N-body simulations. The study of physical systems by particle simulation is called the many-body or the N-body problem[4,7]. N-body simulations have been used to study a wide variety of astrophysical systems during the past years, ranging from small clusters of stars to galaxies and the formation of large-scale structure in the universe. Such studies are conducted in celestial mechanics, plasma physics, and fluid mechanics, as well as in semiconductor device simulation.

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[2,4,6,7,9]. This application was ported to an Intel Paragon and Cray T3D and measurements were collected to study overhead and scalability.

Target Systems

The JPL/ESS Intel Paragon and Cray T3D were used to conduct this study. The Intel Paragon has a total of 64 nodes organized into a 16×4 mesh of which 54 are compute nodes and 8 are service nodes[11]. 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 that are supported by NX library routines for communication and synchronization purposes. The JPL Intel Paragon is operated as applications development platform, with interactive access to all of the compute nodes.

The Cray T3D is a MIMD system with physically distributed but globally addressed memory. The JPL Cray 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[5]. 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 address space. The Alpha processor has a direct-mapped data cache organized into 256 lines with 32bytes per line. Programs can invalidate the local cache as needed to maintain the coherence. 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 inter-processor 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 ap-

plications 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 is an independent computer, one can also use other programming models. One example is the "manager-work" model, in which a "manager" program starts up several "worker" programs on other nodes, then gathers and interprets their results.

In our implementations, the N-body used the manager-worker model. With this model, the manager creates the tree where all spatial information about all particles is 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 that is to be used in the next time-step.

N-Body Problem and the Barnes-Hut Method

The general N-body problem may be stated as the following set of ordinary differential equations[13]:

$$dx_i / dt = v_i \tag{1}$$

$$m_i \,.\, dv_i / dt = \sum_{j \neq i} F_{ij} \tag{2}$$

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

$$F_{ij} = (G \cdot m_i \cdot m_j \cdot r_{ij}) / | r_{ij} |^3 \tag{3}$$

Where G is the universal gravitational constant, m_i and m_j are the masses of particles i and j, and r_{ij} is the position vector separating them.

The gravitational force is "long-range" meaning that there is no cut-off 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[13].

The Barnes-Hut (BH) algorithm[3] is one of a number of algorithm[2,3,6,7,9] that uses 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 Fig. 1, the force on point x_i may be evaluated approximately as:

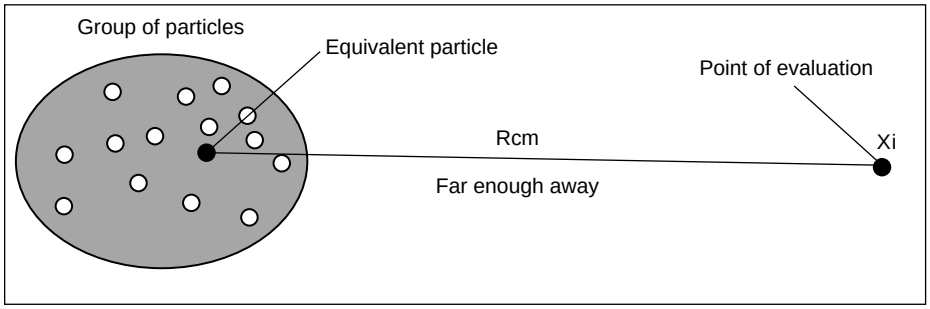

FIG. 1. Approximation of particles by a single point mass.

$$F_i = \sum_j \{(G \cdot m_i \cdot m_j \cdot r_{ij}) / | r_{ij} |^3 \} \approx (G \cdot m_i \cdot M \cdot R_{cm}) / | R_{cm} |^3 \qquad (4)$$

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 oct-tree 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 Fig. 2.

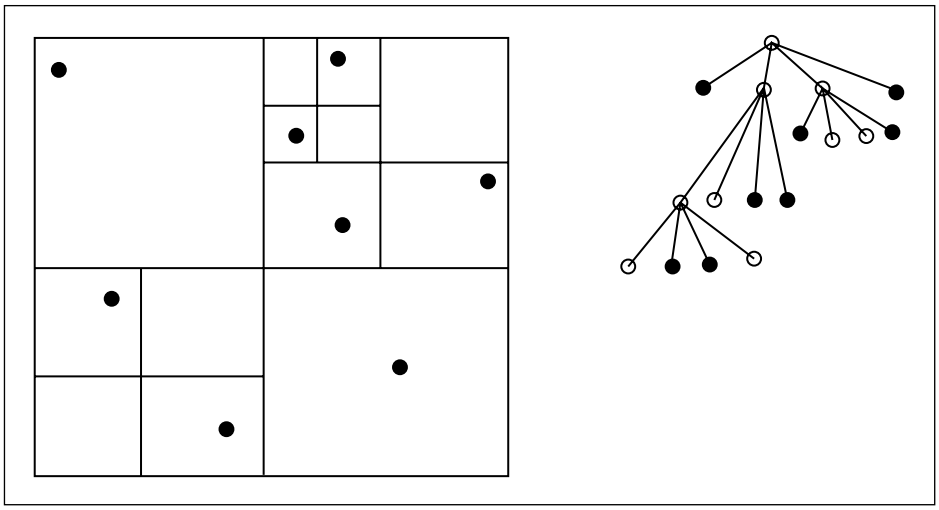

FIG. 2. A 2-D particle distribution and its quadtree.

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 be approximated in unit time. The resulting algorithm, when applied to all bodies, requires $O(\mathrm{N}\log\mathrm{N})$[13] operations to evaluate the forces on all N bodies.

In the BH method, the force-computation phase within a time-step is expanded into four 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 $O(\mathrm{N})$[14].

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[1]. 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 of 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[14]. 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)[13,15].

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 of 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 in order 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 on more need for inter-processor 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 round-off error which results from the non-associatively of floating point operations.

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 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 processor used for the computation. A communication transaction is measured from the point of initiating the communication system call, till the call returns. Imbalance overhead was obtained by averaging the time difference between the completion time at each processor and the minimum completion time over all processors. 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. All timing measurements were wall-clock timing and timed activities were selected such that no global knowledge of time was needed.

Measurements on the Intel Paragon

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

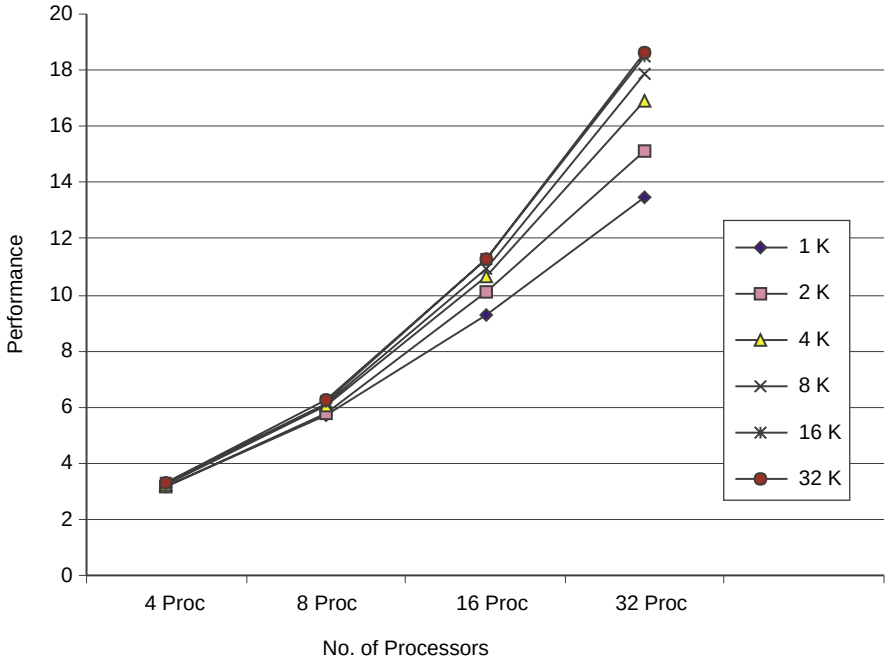

FIG. 3. Scalability of the N-body on the Intel Paragon.

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 manger 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, 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 an 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 increases, most of his overhead is amortized nicely, Fig. 4 to 6. This is due to the rapid growth in the parallel part of the other had, has been minimal in all cases.

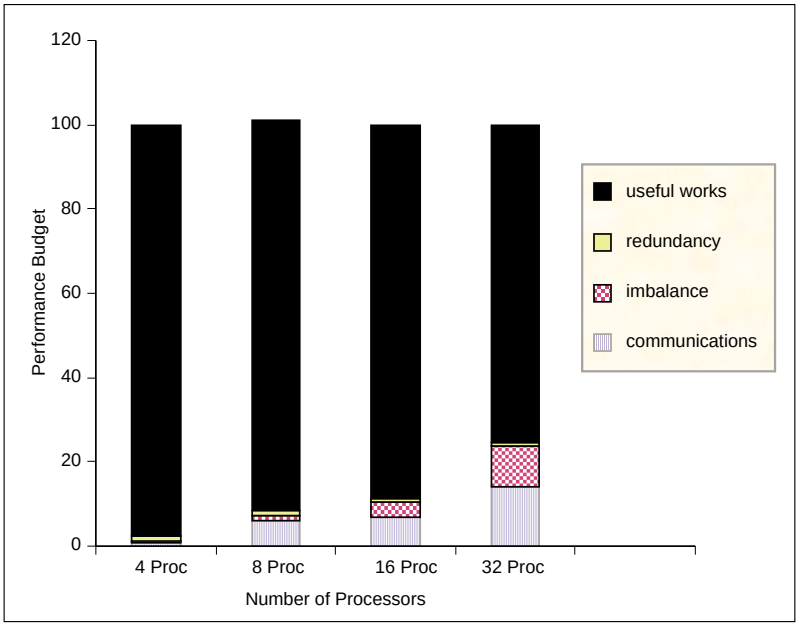

FIG. 4. N-body performance budget at 1k bodies.

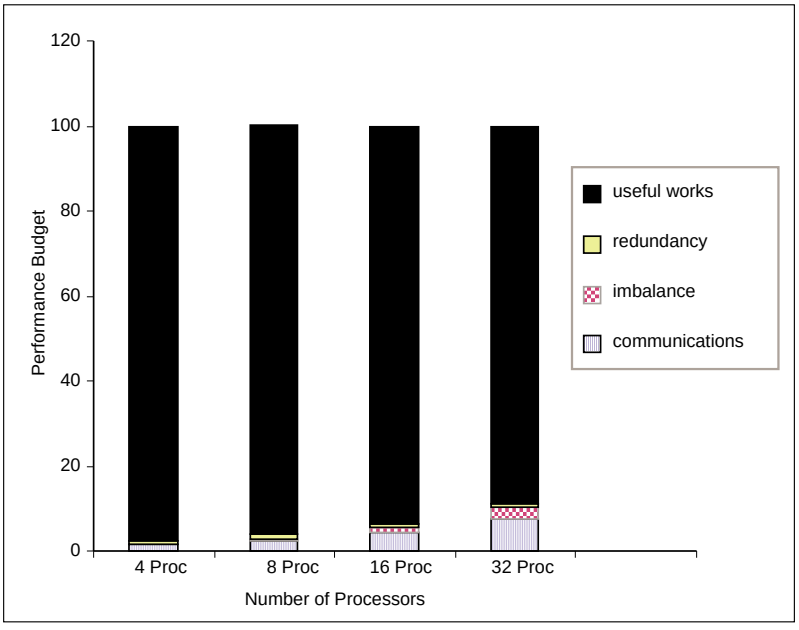

FIG. 5. N-body performance budget at 4k bodies.

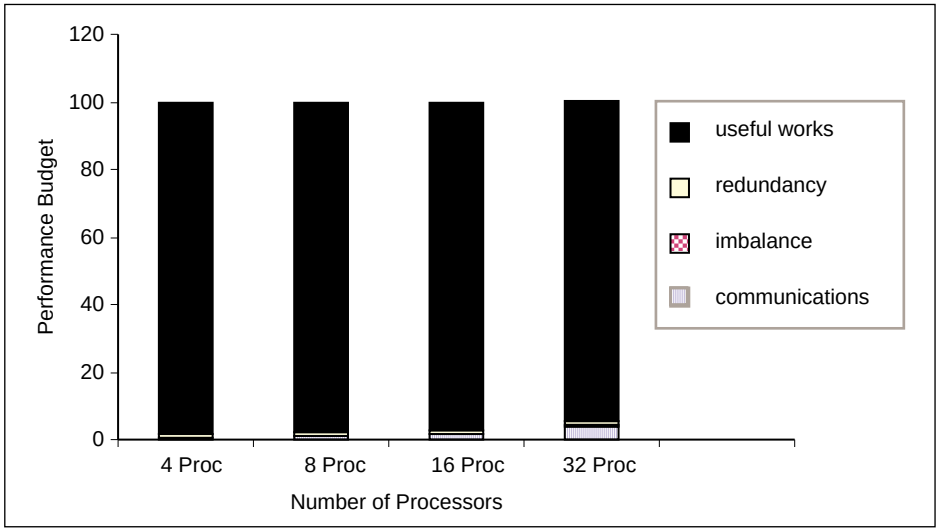

FIG. 6. N-body performance budget at 32k bodies.

Measurements on the Cray T3D

On the Cray T3D, N-Body simulation presents a better overall "picture". Although the NX routines are considered to be superior to PVM calls. Clearly, processor and network speed are dominant factors. Figure 7 indicates that the scalability is drawn by the communication component, a fact supported by performance budget figures as well, see Fig. 8, 9 and 10. When the ratio of the data to be processed to the number of processing nodes to be communicated is large enough, the code scales quite well, see Fig. 7. Communication time exhibits a smooth slope with increasing data sizes. Consequently, the performance budget figures include more portions of useful work than ones on the Intel Paragon. Characteristically, the execution across the processors are again well-balanced and redundantly is negligibly small.

Observations

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.

Effect of Programming Model: The use of the manager-worker model in the N-body code has resulted in many experiences. 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

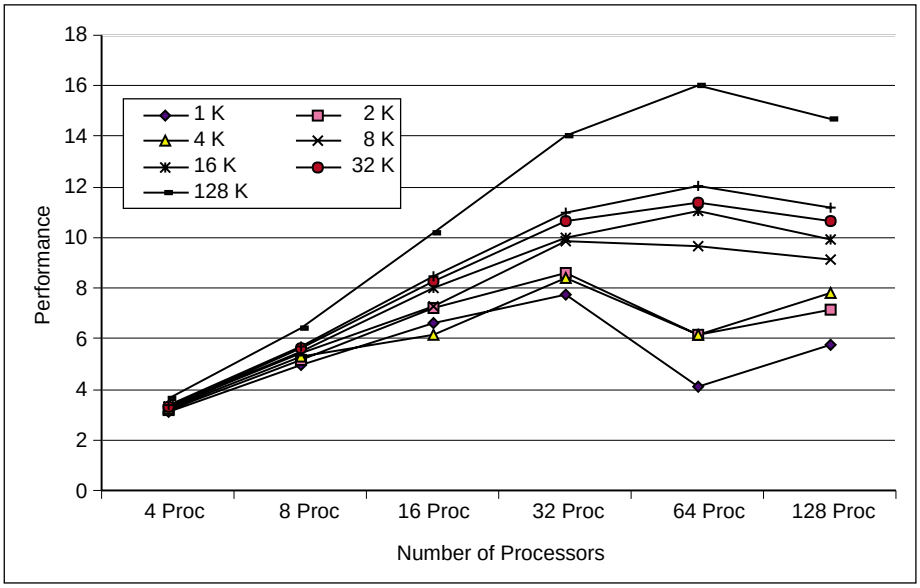

FIG. 7. Scalability of the N-body on the Cray T3D.

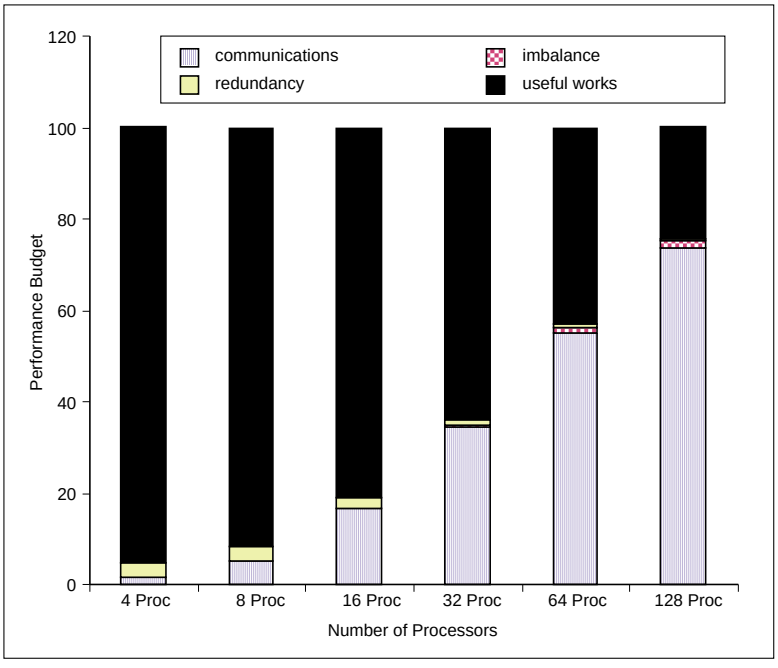

FIG. 8. N-body performance budget at 1k bodies.

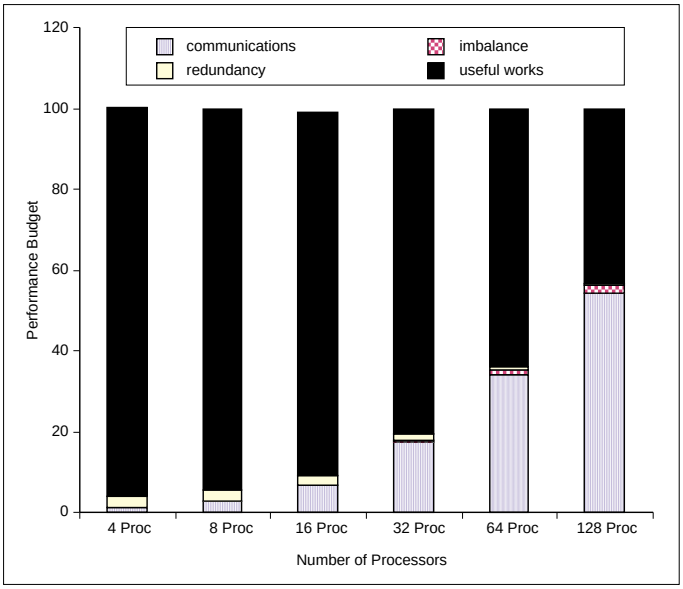

FIG. 9. N-body performance budget at 4k bodies.

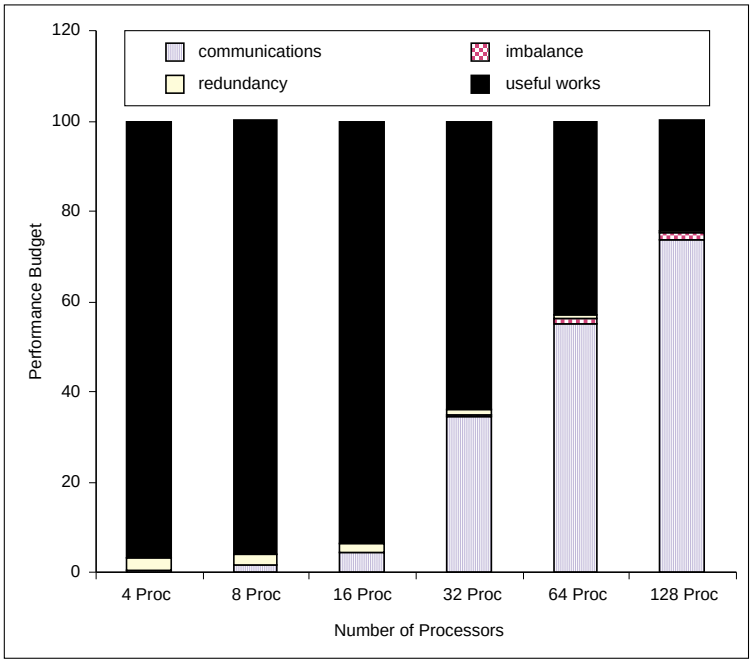

FIG. 10. N-body performance budget at 32k bodies.

focal point of communication created by the manager and the variability of the communication distances from arbitrary nodes to the manager.

Effect of Memory Management: Again, super-linear 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 that takes such memory-related factors into account.

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 type of overhead can become more dominant than others. In fact, in many cases, reducing one type of overhead. For example, in many cases communications can be replaced by redundancy and vise versa. A general rule, however, is that redundancy is cheaper than communications, in most cases.

Physical Effects: One phenomenon that was observed in the Intel 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 are maintained. After repeated measurements and investigations, it was found that processors that are physically closer to the cooling system tend to run slower that those that are farther away[10]. Up to 7% variability in exertion time was observed and attributed to this phenomena.

Conclusion

In this study some of the sources of overhead were identified and measured for a real application 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 unprocessed 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.

Acknowledgment

I would like to acknowledge the King Fahd University of Petroleum and Minerals for their support. This work has been supported by the Center of Ex-

cellence in Space Data and Information Sciences at NASA Goddard Space Flight Center under Grant No. NAS5-30428. This work has been supported by NASA High-Performance Computing and Communications (HPCC) program through CESDIS/USRA, Grant No. NAS5-30428.

References

[1] **Almojel, A.I.,** "The Implementation and Performance Evaluation of N-body Gravitational Simulation Algorithm on High-Performance Computers", *J. Computers & Electrical Engineering,* vol. **26,** no. 3-4, pp. 297-316, April (2000).

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

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

[4] **Birdsall, C.K.** and **Langton, A.B.,** *Plasma Physics Via Computer Simulation,* McGraw-Hill Inc., New York, (1985).

[5] *Cray T3D User's Guide,* (1994).

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

[7] **Hockney, R.W.** and **Eastwood, J.W.,** *Computer Simulation Using Particles,* Adam Hilger, (1988).

[8] http://ess.jpl.nasa.gov.

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

[10] **Meajil, A.I.,** "An Implementation of a Tree-Based N-Body Algorithm on Message-Passing Architectures", Applied Parallel Computing: Industrial Computation and Optimization, *Lecture Notes in Computer Science* (1184), J. Wasniewski, *et al.* (Eds.), Springer, pp. 504-514, September (1996).

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

[12] **Poo, G.S.** and **Goscinski, A.M.,** "Introduction to Special Issue on Network-Based Parallel and Distributed Computing", *Journal of Computer Communications,* vol. **22,** no. 11, p. 987, July (1999).

[13] **Salmon, J.,** *"Parallel N Log N N-Body Algorithms and Applications to Astrophysics",* COMPCON, Spring (1991).

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

[15] **Zhiling, L., Valerie, T.** and **Grey, B.,** "A Novel Dynamic Load Balancing Scheme for Parallel Systems", *J. Parallel and Distributed Computing,* vol. **62,** no. 12, pp. 1763-1781, December (2002).

تقييم موازنة لمدى قابلية التوسع والترقية وإمكانية القياس (المعايرة) للأنظمة الحاسوبية ذات الأداء العالي

عبدالله إبراهيم المعجل

قسم هندسة الحاسب الآلي ، جامعة الملك فهد للبترول والمعادن

الظهـــران – المملكة العربية السعودية

المستـخلص. إن الأنظمـة الحسـابية ذات الأداء العـالمي HPCS والتي تعتـمد على عـمليـة المعـالجة المتـوازية لديهـا قـدرة على تلبيـة المتطلبات الحسابية التي تميزت بالنمو السريع ، وذلك بتكلفة اقتصادية . فكما أن تلك الآلات والأجهزة تكبر في حجمها ، فإن إجمالي التوازي ينمو بطريقة يمكن أن تحد من إمكانية القياس والجدولة (المعايرة) . وهذا البحث يلقي بعض الضوء على مصادر وديناميكيات وأحجام بنود المصروفات المختلفة وتأثيرها على الأداء . يمكن الحصول على النتائج من خلال القياسات التجاربية للتطبيقات التي أجرتها وكالة ناسا لعلوم الفضاء على الأنظمة الحسابية ذات الأداء العالمي HPC في معمل الدفع النفاث / علوم الأرض والفضاء .