

USING AMDAHL'S LAW AS A METRIC TO DRIVE CODE PARALLELIZATION: TWO CASE STUDIES

Mihai Horoi

DEPARTMENT OF PHYSICS, CENTRAL MICHIGAN UNIVERSITY, U.S.A.

Richard J. Enbody

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, MICHIGAN STATE UNIVERSITY, U.S.A.

Summary

Using Amdahl's law as a metric, the authors illustrate a technique for developing efficient code on massively parallel processor (MPP) performance class networks to solve nontrivial, high performance scientific problems. They also show the importance of collective communication within the message-passing interface (MPI) paradigm for some applications. Given the popularity of Beowulf-like clusters of workstations, this work also indicates the necessity of a scalable high performance network for obtaining efficient performance in parallel code. Using this approach, the authors were able to obtain an effective speedup (comparison with the best sequential time) of 170 when using 256 of the Cray T3E 900 processing elements (PEs) to solve a carbon, molecular-dynamic problem. The authors also examine the approach on a very different application: a Lanczos eigenvalue solver.

Address reprint requests to Richard J. Enbody, Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, U.S.A.; e-mail: enbody@cse.msu.edu.

The International Journal of High Performance Computing Applications,
Volume 15, No. 1, Spring 2001, pp. 36-41
© 2001 Sage Publications, Inc.

Creating efficient parallel codes is a complex process that does not appear to have a solution in sight (Kuck, 1996). In this paper, we illustrate how one can use Amdahl's law to analyze the characteristics of a program to achieve significant speedup. One can extend Amdahl's law to incorporate the most important factors limiting performance (Horoi and Enbody, 1998):

- the magnitude of the sequential fraction of the code,
- the communication overhead, and
- the work imbalance on different processors.

This report describes the quantitative analysis of these factors and the solutions used to diminish or eliminate their contribution in two case studies.

Using this approach, we were able to obtain an *effective speedup* (comparison with the best sequential time) of 170 when using 256 of the Cray 3E 900 processing elements (PEs).

To understand the factors responsible for the limited performance of parallel codes, we turn to Amdahl's law (Amdahl, 1967). Extending Amdahl, the speedup S can be given by

$$S = \frac{T_1}{\frac{\gamma T_1}{\bar{\eta} P} + (1 - \gamma)T_1 + T_{comm}}, \quad (1)$$

where T_1 represents the sequential (1PE) time, P is the number of PEs, γ represents the fraction of code that can be fully parallelized ($1 - \gamma$ is the sequential fraction), $\bar{\eta}$ describes the average imbalance of the load on each PE, and T_{comm} represents the total communication overhead. The average imbalance $\bar{\eta}$ can be defined in terms of the imbalance between two synchronization points in the parallel fraction of the code. It has values between $1/P$ (completely unbalanced: all workload taken by a single PE) and 1 (completely balanced between PEs). It is well known that as P increases, the sequential portion $(1 - \gamma)T_1$ can dominate even for relatively large values of γ . For example, code that is 90% parallel ($\gamma = .9$), is perfectly balanced ($\bar{\eta} = 1$), and has no communication ($T_{comm} = 0$) can only achieve a maximum speedup of 10 for any number of processors.

Our methodology is as follows:

- use equation (1) to identify the three performance factors that one can tune: γ , T_{comm} , and $\bar{\eta}$;
- for each term, isolate and measure its individual con-

- using these metrics, tune the code to improve performance.

The remainder of this report describes the quantitative analysis of these factors and the solutions used to diminish or eliminate their effects in two case studies.

The first case study is a code that simulates the self-assembly of carbon nanostructures from individual atoms developed at the Center for Fundamental Materials Research at Michigan State University. Precise interatomic forces are calculated using a linear combination of atomic orbitals (LCAO) technique that is based on the *ab initio* local density functional formalism (Lee, Kim, and Tománek, 1997). The many-body LCAO Hamiltonian, combined with the recursion technique (Zhong, Tománek, and Bertsch, 1993), allows a semianalytical evaluation of forces acting on individual atoms that scales linearly with the number particles. Even though we use carbon atoms, the method is quite general and could be used with any other atomic species forming covalent bonds. To calculate the force acting on one atom, the contribution from the electrons belonging to the neighbors is considered and, as a second-order effect, the contribution of all neighbors of the first layer of neighbors. The “neighbors-of-neighbors” contribution adds complexity to the communication protocol. The required synchronization prevents the code from being “embarrassingly parallel.”

The sequential code has 5400 lines and 64 subroutines written in FORTRAN 77. The code must synchronize each time the forces are calculated, and results from each processor must be known to all other processors. At each time step, it calculates the forces acting on each atom, integrates the equation of motions using the fourth-order Runge-Kutta method, and uses the new coordinates to calculate the new forces acting on each atom.

Based on research at Sandia National Labs (Plimpton, 1995), we began with a spatial decomposition of the problem but found that the neighbors-of-neighbors characteristic of the code created too much communication and was time-consuming to load balance (Enbody, Purdy, and Severance, 1995; Severance and Enbody, 1994). We turned to particle decomposition, the assignment of a fixed number of particles to processors, since it can be satisfactorily statically load balanced for the relatively homogeneous calculations of interest to us and did not increase our communication load significantly.

In this study, we use three nanotube configurations: (C480), a 480-carbon atom nanotube at 350°K; (C512), a

512-carbon atom nanotube at 0°K; and (C1024), a 1024-carbon atom nanotube at 0°K. For the purposes of determining speedup, we only calculate over a few times which is a worst case since startup costs are not amortized. To calculate the speedup, we always compare the wallclock time given by the parallel version of the code with the best 1PE time given by the original sequential code.

Let us now consider the three tunable parts of Amdahl’s law: γ (parallel portion), T_{comm} (communication), and $\bar{\eta}$ (balance). Beginning with communication, it is difficult to isolate T_{comm} for quantitative measurement on real code. We used a master-slave approach in parallelizing our code, so we wrote a synthetic code that performed the broadcasting of data from master to slaves and gathering back partial data from slaves to master without doing any force calculations. We considered two message-passing libraries, PVM and MPI, and two different network architectures, Cray T3E and IBM SP2. Assuming that the busiest communication link is the master, we measured the *effective* throughput (total amount of data received and sent back by all slaves in unit of time) at that link. We plotted PEs versus throughput and found that using MPI on the Cray showed communication throughput growing nearly linearly up to the 256 PEs available. However, PVM on the Cray and MPI on the IBM stopped improving at around 25 PEs, resulting in the MPI on the Cray having at least eight times the throughput. The efficient implementation of MPI’s collective communication on the Cray appeared to be the significant factor for our master-slave approach.

To measure the sequential fraction of the code, $1 - \gamma$, we measured the total time used for calculation, T_{tot} , and the time spent in the loop containing the load shared among the PEs. Pseudocode describing that portion of the code is presented in Figure 1.

The results of these measurements are in Table 1. One can observe that the sequential fraction of the code is in the range 1% to 5%. (The $T_{cluster1}$ and $(1 - \gamma)$ columns will be discussed later.) Even though the sequential fraction of the code does not account for more than 1% in the configuration C480, Amdahl’s law predicts a maximum ideal speedup ($\bar{\eta}=1$, $T_{comm} = 0$) of 59 when 128 PEs are used. This factor is expected to further limit the performance of the parallel code for configurations with more atoms, such as C512 and C1024, due to the $O(N^2)$ number of operations involved in the **cluster1** routine.

To reduce the effect of the sequential fraction of code, we further analyzed possible contributors to that part. We found that the call to the **cluster1** routine (see Figure 1) is

```

subroutine force
  call broadcast_coordinates
  call cluster1 ( n_atoms )
  do iatom = atom_start, atom_end
    call cluster2 ( iatom )
    call calculate_force
  enddo
  call gather_results

```

Fig. 1 Pseudocode of the parallelized portion of the code before optimization

**Table 1
Sequential Component Code before $(1 - \gamma)$ and after $(1 - \gamma)$ the Optimization**

N	Before Optimization					After
	T_{tot}	T_{loop}	ΔT	$1 - \gamma$	$T_{cluster1}$	$1 - \gamma'$
480	1850	1833	17	0.009	16	0.0005
512	1304	1276	28	0.022	28	0.001
1K	2672	2560	112	0.042	112	0.001

“One can observe that the most important factor influencing the imbalance is the degree of incommensurability of the number of PEs with the number of atoms (i.e., the result of the modulus division of the number of atoms to the number of PEs). It is not a startling result, but the analysis makes it obvious.”

responsible for the largest contribution to the sequential fraction of the code (Table 1). This routine calculates the first layer of neighbors of all atoms. The **cluster2** routine, calculating the second layer of neighbors for each atom (neighbors-of-neighbors), appeared in the loop in the original sequential code. It uses the results saved by the **cluster1** routine and is able to save approximately 10% of the total time in the original sequential code. Having **cluster1** outside the loop was a valuable optimization for the sequential code, but it hurts the parallel implementation.

Our strategy was to move the calculation done by the **cluster1** routine into the *do* loop, allowing calculation of both layers of neighbors in a scalable way. The new code is slower by approximately 11% for 1 PE ($T_1' \approx 1.11T_1$), but it scales very well when the number of PEs is increased. A simulation to quantify the remaining sequential fraction of code shows a decrease of $1 - \gamma'$ to less than 0.1% (see last column in Table 1).

Last, we measure the contribution of the average imbalance $\bar{\eta}$ using a skeleton of the code. The imbalance factor η for different calls of the force routine can be defined as the ratio of the total workload between two synchronization points in the parallel portion of the code (a call to the force routine in our example) and P (number of PEs) times the maximum workload assigned to a single PE for the same calculation. If the total workload can be evenly divided to all PEs, η will be 1. We plotted PEs versus η and observed that for 2 PEs and an even number of particles, one observes a small imbalance due to a slightly different workload for different atoms, while for 128 PEs, the decrease in imbalance factor is significant in the C480 case. At the root of the imbalance is the difficulty of equally distributing a relatively small number of atoms on a relatively large number of PEs available to us. One can observe that the most important factor influencing the imbalance is the degree of incommensurability of the number of PEs with the number of atoms (i.e., the result of the modulus division of the number of atoms to the number of PEs). It is not a startling result, but the analysis makes it obvious.

We tested the combined effect of these modifications based on analysis of γ , T_{comm} , and $\bar{\eta}$. Figure 2 displays the effective speedup versus PEs. The effective speedup is defined with respect to the best 1 PE time given by the original sequential code. We also include the lines denoted by “x” and “0.5*x,” which describe the region of high efficiency defined by Kuck (1996). The results for the

512-atom (MPI512) and 1024-atom (MPI1024) configurations scale very well: the effective speedups lying in Kuck's region of high efficiency. A maximum effective speedup of 170 was obtained for the 1024-atom configuration using 256 PEs. The original version before modifications (labeled PVM512) is included for comparison.

We also tested the improvements individually. None of the improvements individually could keep the speedup within Kuck's (1996) high efficiency region.

Our second case study parallelizes a Lanczos eigenvalue solver subroutine for real symmetric matrices (Lanczos, 1950). We are interested in applications in which the matrix is essentially nonsparse and the lowest (highest) eigenvalues are needed. This approach is of great interest for many-body quantum calculations, such as in nuclear shell model calculations (Horoi, Volya, and Zelevinsky, 1999). These calculations require the first 8 to 10 eigenvalues for matrices of order 10,000 to 100,000. The sparsity is usually 0.5.

The subroutine structure presented above indicates the matrix multiplication subroutine HMULT, which is an $O(N^2)$ operation, as the best target for parallelization. The natural approach is to share the nonzero matrix elements between all PEs; this can be achieved by a simple broadcast operation before the iterative process starts. As a consequence of the large number of nonzero matrix elements (of the order of 10^7 for matrix dimensions of the order 10,000) and a relatively small number of PEs (256-512), the workload decomposition can be easily balanced ($\bar{\eta}$ in equation (1) will be very close to unity). In this case, the Amdahl metric indicates that one should concentrate on the effect of the sequential portion of the code, γ , and of the communication overhead T_{comm} .

```

SUBROUTINE LANZOS
.
INITIALIZE
ITER = 3
1 CONTINUE
ITER = ITER + 1
CALL HMULT ! Calculate Lanczos basis:
CALL HMULT ! Calculate the eigenvalues of
CALL EIGENVALUES ( ITER ) ! the tridiagonal matrix.

TEST CONVERGENCY IN EIGENVALUES
IF FALSE GO TO 1
ELSE
DO I = 1, NEIGENVALUES
CALL EIGENVECTOR ( I )
CALCULATE EIGENVECTOR I OF THE ORIGINAL MATRIX
ENDDO
END

```

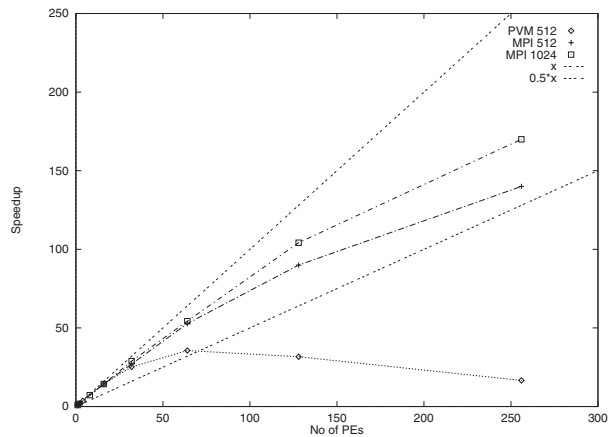


Fig. 2 Final speedup incorporating all modification versus original PVM

“As a consequence of the large number of nonzero matrix elements (of the order of 10^7 for matrix dimensions of the order 10,000) and a relatively small number of PEs (256-512), the workload decomposition can be easily balanced.”

Table 2
Instrumentation Results for the Lanczos
Eigenvalue Solver

N	b	NEIG	T_{TOT}	T_{HMULT}	ΔT	$1 - \gamma$ (%)
1024	256	8	6.13	5.86	0.27	4.4
2048	512	8	25.01	24.54	0.48	1.9
4096	1024	8	73.82	72.85	0.96	1.3

“We found that using Amdahl’s law as a metric was effective for the analysis and improvement in the performance of parallel code.”

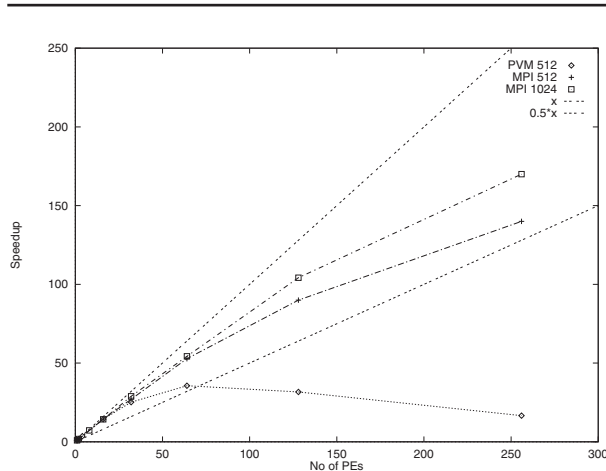


Fig. 3 Speedup for different matrix dimensions

Analysis of γ yields Table 2 (similar to Table 1), which presents the results of timing the code for a banded matrix of sparsity ~ 0.5 . One can see the decrease of the sequential fraction of the code, $1 - \gamma$, when the matrix dimension is increased. Since we are most interested in large dimensions, the Amdahl metric indicates that γ works in our favor for this problem. Remember that Amdahl’s law (Horoi and Enbody, 1998) requires a sequential fraction smaller than 1% to obtain scalability beyond 100 PEs.

We already showed above that if the communication overhead for a small number of PEs (4 to 8) is small, due to the scaling of our MPI broadcasting-gathering approach on the Cray T3E architecture, the communication overhead for a larger number of PEs will not affect the performance of the parallel code. The result is that our previous analysis of broadcast gathering with respect to T_{comm} applies directly to this problem.

Figure 3 shows the speedup obtained on the Cray T3E at Pittsburgh Supercomputer Center for different matrix dimensions. The x and $0.5x$ again define Kuck’s region of high efficiency. For dimensions larger than 8000, the code scales up very efficiently. We conclude that the analysis based on Amdahl’s three-factor formula is also a very powerful instrumentation tool for the design of a very efficient parallel Lanczos eigenvalue solver routine.

In conclusion, we found that using Amdahl’s law as a metric was effective for the analysis and improvement in the performance of parallel code. It steered our parallelization by helping us identify and reduce three important components of speedup: γ , T_{comm} , and $\bar{\eta}$. In our first case study, we obtained a speedup of 170 on 128 Cray T3E PEs when integrating the equations of motions for 1024 particles. In our second case, our Lanczos eigenvalue solver routine for the case of essentially nonsparse matrices using Amdahl’s law was shown to scale well up to 128 processing elements (PEs) for matrix dimensions of 8000. From a computational point of view, the techniques we developed for this project have the potential for application to any type of code that involves the broadcasting and gathering of data with synchronization.

ACKNOWLEDGMENTS

This research was supported in part by grant PHY960002P from the Pittsburgh Supercomputing Center and by the Office of Naval Research, grant N00014-99-1-0252.

BIOGRAPHIES

Mihai Horoi received his B.A. and M.S. in physics from University of Bucharest, Romania, in 1979 and his Ph.D. from the In-

stitute of Atomic Physics in Bucharest. He obtained his M.S. degree in computer science from Michigan State University in 1997. After spending his Alexander von Humboldt fellowship at Giessen University in Germany, he conducted postdoctoral research at the National Superconducting Cyclotron Laboratory in East Lansing. He joined the Physics Department at Central Michigan University in 1995.

Richard J. Enbody is an associate professor in the Department of Computer Science and Engineering. He joined the faculty in 1987 after earning his Ph.D. in computer science from the University of Minnesota. He received his B.A. in mathematics from Carleton College in Northfield, Minnesota, in 1976 and spent six years teaching high school mathematics in Vermont and New Hampshire. His research interests are in computer architecture, Web-based distance education, and parallel processing, especially the application of parallel processing to computational science problems.

REFERENCES

- Amdahl, G. M. 1967. Validity of single processor approach to achieving large-scale computing capability. In *Proc. AFIPS Conf.*, Reston, VA.
- Enbody, R., R. Purdy, and C. Severance. 1995. Dynamic load balancing. In *Seventh SIAM Conference on Parallel Processing for Scientific Computing*, February, pp. 645-646.
- Horoi, M., and R. Enbody. 1998. Performance analysis and optimization of a parallel carbon molecular dynamic code on a Cray T3E. In *International Conference on Parallel Processing*, August, p. 62.
- Horoi, M., A. Volya, and V. Zelevinsky. 1999. Chaotic wave functions and exponential convergence of low-lying energy eigenvalues. *Phys. Rev. Lett.* 80:2064-67.
- Kuck, D. J. 1996. *High performance computing*. Oxford, UK: Oxford University Press.
- Lanczos, C. 1950. *ARTICLE TITLE?*. *J. Res. Natl. Bur. Stand.* 45:255.
- Lee, Y. H., S. G. Kim, and D. Tománek. 1997. *ARTICLE TITLE?*. *Phys. Rev. Lett.* 78:2393.
- Plimpton, S. J. 1995. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.* 117:1-19.
- Severance, C., and R. Enbody. 1994. Evolving dynamic load balancing to shared memory parallel processors. *Supercomputing*, November.
- Zhong, W., D. Tománek, and G. F. Bertsch. 1993. *ARTICLE TITLE?*. *Solid State Comm.* 86:607.