An Examination of the Performance of Parallel Calculation of the Radiation Integral on a Beowulf-Class Computer

Daniel S. Katz, Tom Cwik, Thomas Sterling

Jet Propulsion Laboratory

California Institute of Technology

Abstract – This paper uses the parallel calculation of the radiation integral for examination of performance and compiler issues on a Beowulf-class computer. This type of computer, built from mass-market, commodity, of-the-shelf components, has limited communications performance and therefore also has a limited regime of codes for which it is suitable. This paper first shows that this code falls within this regime, and then examines performance data, including run-time, scaling, compiler choices, and the use of some hand-tuned optimizations, comparing results from a Beowulf and a Cray T3D.

Keywords – beowulf, cluster, pile of PCs, parallel computation, physical optics, radiation integral

Contact information – Daniel S. Katz, 4800 Oak Grove Drive, MS 168-522, Pasadena, CA, 91109-8099, Daniel.S.Katz@jpl.nasa.gov, ph: 1.818.354.7359, fax: 1.818.393.3134

1. Introduction

A typical small Beowulf system, such as the machine at the Jet Propulsion Laboratory (JPL) may consist of 16 nodes interconnected by 100Base-T Fast Ethernet. Each node may include a single Intel Pentium Pro 200 MHz microprocessor, 128 MBytes of DRAM, 2.5 GBytes of IDE disk, and PCI bus backplane, and an assortment of other devices. At least one node will have a video card, monitor, keyboard, CD-ROM, floppy drive, and so forth. But the technology is evolving so fast and price performance and price feature curves are changing so fast that no two Beowulfs ever look exactly alike. Of course, this is also because the pieces are almost always acquired from a mix of vendors and distributors. The power of *de facto* standards for interoperability of subsystems has generated an open market that provides a wealth of choices for customizing one's own version of Beowulf, or just maximizing cost advantage as prices fluctuate among sources. Such a system will run the Linux [1] operating system freely available over the net or in low-cost and convenient CD-ROM distributions. In addition, publicly available parallel processing libraries such as MPI [2] and PVM [3] are used to harness the power of parallelism for large application programs. A Beowulf system such as described here, taking advantage of appropriate discounts, costs about $30K including all incidental components such as low cost packaging.

At this time, there is no clearly typical medium to large Beowulf system, since as the number of processors grows, the choice of communications network is no longer as clear. (If the machine can use a crossbar that can support the entire machine, the choice is simply to use that crossbar switch.) Many choices exist of

various topologies of small and large switches and hubs, and combinations thereof.

Naegling, the Beowulf-class system at the California Institute of Technology, which currently has 140 nodes, has had a number of communications networks. The first was a tree of 8- and 16-port hubs. At the top of the tree was a standard 100 Mbit/s 16-port crossbar, with full backplane bandwidth. Each port of this was connected to a hub. Each hub had 100 Mbit/s ports connected to 8 or 16 computers; however, the backplane bandwidth of each hub was also 100 Mbit/s. The second topology used additional 16-port crossbars at the low level of the tree, where 15 ports of each crossbar were connected to computers, and the last port was connected to a high-level crossbar. A third network (which is not functioning currently) involves 2 80-port switches, connected by 4 Gbit/s links. Each switch has 100 Mbit/s ports and full backplane bandwidth.

The Beowulf approach represents a new business model for acquiring computational capabilities. It complements rather than competes with the more conventional vendor-centric systems-supplier approach. Beowulf is not for everyone. Any site that would include a Beowulf cluster should have a systems administrator already involved in supporting the network of workstations and PCs that inhabit the workers' desks. Beowulf is a parallel computer, and as such, the site must be willing to run parallel programs, either developed in-house or acquired from others. Beowulf is a loosely coupled, distributed memory system, running message-passing parallel programs that do not assume a shared memory space across processors. Its long latencies require a favorable balance of computation to communication and code written to balance the workload across processing nodes. Within the constrained regime in which Beowulf is

appropriate, it should provide the best performance to cost and often comparable performance per node to vendor offerings.

2. Physical Optics

The code described in this paper [4] is used to design and analyze reflector antennas and telescope systems. It is based simply on a discrete approximation of the radiation integral [5]. This calculation replaces the actual reflector surface with a triangularly faceted representation so that the reflector resembles a geodesic dome. The Physical Optics (PO) current is assumed to be constant in magnitude and phase over each facet so the radiation integral is reduced to a simple summation. This program has proven to be surprisingly robust and useful for the analysis of arbitrary reflectors, particularly when the near-field is desired and the surface derivatives are not known.

Because of its simplicity, the algorithm has proven to be extremely easy to adapt to the parallel computing architecture of a modest number of large-grain computing elements. The code was initially parallelized on the Intel Paragon, and has since been ported to the Cray T3D, T3E, and Beowulf architectures.

For generality, the code considers a dual-reflector calculation, as illustrated in Figure 1, which can be thought of as three sequential operations: (1) computing the currents on the first (sub-) reflector using the standard PO approximation; (2) computing the currents on the second (main) reflector by utilizing the currents on the first (sub-) reflector as the field generator; and (3) computing the required

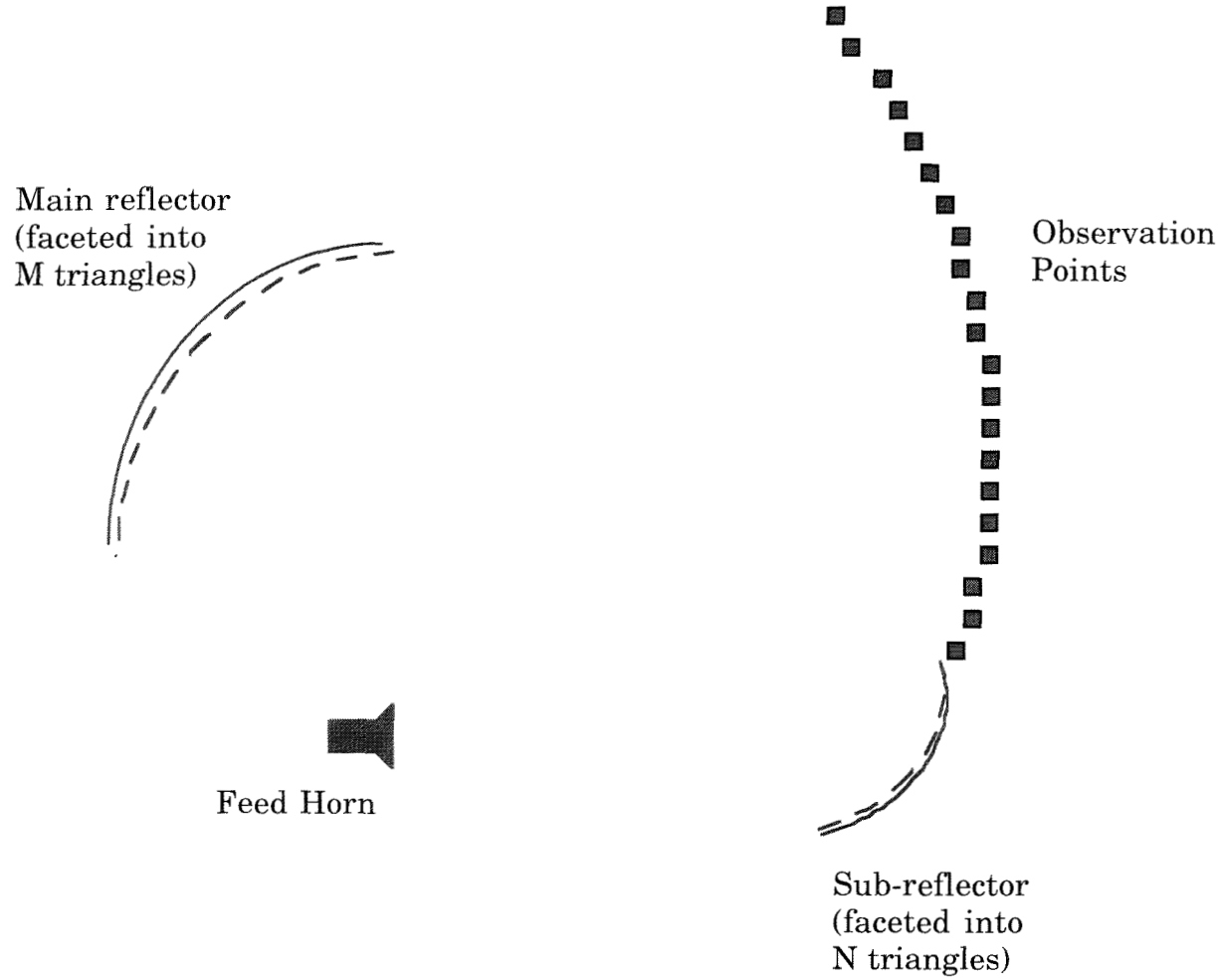

Figure 1. The dual reflector Physical Optics problem, showing the source, the two reflectors, and the observation points.

observed field values by summing the fields from the currents on the second (main) reflector. The most time-consuming part of the calculation is the computation of currents on the second reflector due to the currents on the first, since for N triangles on the first reflector, each of the M triangles on the second reflector require an N-element sum over the first. At this time, the code has been parallelized by distributing the M triangles on the second reflector, and having all processors store all the currents on the N triangles of the first reflector (though

the computation of the currents of the first reflector is done in parallel.) Also, the calculation of observed field data has been parallelized. So, the three steps listed above are all performed in parallel. There are also sequential operations involved, such as I/O and the triangulation of the reflector surfaces, some of which potentially could be performed in parallel, but this would require a serious effort, and has not been done at this time.

This code is written in FORTRAN, and has been parallelized using MPI. Communication is required in two locations of the code. At the end of the first step, after each processor has computed a portion of the currents on the first reflector, the currents must be broadcast to all the processors. While this may be done in many ways, a call to *MPI_Allgatherv* is currently used. During the third step, each processor calculates a partial value for each final observed field, by integrating over the main reflector currents local to that processor. A global sum (an *MPI_Reduce* call) is required to compute the complete result for each observed field value. Since there are normally a number of far fields computed, currently there are that number of global sums. These could be combined into a single global sum of larger length, but this has not been done at this time, since the communication takes up are such a small portion of the overall run time.

3. Results

A note to the reviewers: *The results show below are only for 16 processors, because the communications network on the Beowulf system at Caltech is currently the second network mentioned in section 1 above. It is not possible to achieve decent results for runs on more that one switch, due to the severely limited bandwidth between switches. I expect the third communications network*

to be in place and working before the final version of this paper is due, and I plan to include results on at least 64 processors in the final version.

Tables 1 and 2 show initial timing results for the PO code, for 2 difference size sub-reflectors, with the same size main reflector. Each run is broken down into three parts. Part I is input I/O and triangulation of the main reflector surface, some of which is done in parallel. No communication occurs in part I. Part II is triangulation of the sub-reflector surface (sequential), evaluation of the currents on the sub-reflector (parallel), and evaluation of the currents on the main reflector (parallel). A single *MPI_Allgatherv* occurs in part II. Part III is evaluation of the observed fields (parallel) and I/O (on only one processor). A number of 3 word global sums occur in part III, one for each observation point. In the test cases used here, there are 122 observation points. The Beowulf results are from the 16 node system, using the GNU g77 compiler.

Number of Processors	Beowulf			T3D		
	I	II	III	I	II	III
1	5.10	307	98.2	14.5	399	83.9
4	3.12	77.4	25.7	8.88	100	21.3
16	2.73	20.1	6.47	8.15	26.1	6.26

Table 1. Original timing results (in seconds) for PO code, for M=40,000, N=400.

Number of Processors	Beowulf			T3D		
	I	II	III	I	II	III
1	0.0850	64.3	1.64	0.334	85.8	1.90
4	0.0515	16.2	0.431	0.147	19.8	0.354
16	0.0437	4.18	0.110	0.136	4.99	0.105

Table 2. Original timing results (in minutes) for PO code, for M=40,000, N=4,900.

It may be observed from Tables 1 and 2 that the Beowulf code performs better than the T3D code, both in terms of absolute performance as well as scaling from 1 to 16 processors. The absolute performance difference can be explained by the faster CPU on the Beowulf versus the T3D, and the very simple and limited communication not enabling the T3D's faster network to influence the results. The scaling difference is more a function of I/O, which is both more direct and more simple on the Beowulf, and thus faster. By reducing this part of the sequential time, scaling performance is improved. Another way to look at this is to compare the results in the two tables. Clearly, scaling is better in the large test case, in which I/O is a smaller percentage of overall time.

A second set of Beowulf runs were then made, using a commercial compiler (Absoft f77) on the 16-node Beowulf system. The results from these run are compared with the original Beowulf results in Tables 3 and 4. The change of compilers caused the overall run-times to go down by approximately 30%.

Number of	Compiler 1			Compiler 2		
Processors	I	II	III	I	II	III
1	5.10	307	98.2	3.19	230	56.0
4	3.12	77.4	25.7	1.85	57.7	14.2
16	2.73	20.1	6.47	1.52	14.6	3.86

Table 3. The effect of a second Beowulf compiler, shown by timing results (in seconds) for PO code, for M=40,000, N=400.

Number of Processors	Compiler 1			Compiler 2		
	I	II	III	I	II	III
1	0.0850	64.3	1.64	0.0482	46.4	0.932
4	0.0515	16.2	0.431	0.0303	11.6	0.237
16	0.0437	4.18	0.110	0.0308	2.93	0.0652

Table 4. The effect of a second Beowulf compiler, shown by timing results (in minutes) for PO code, for M=40,000, N=4,900.

It was also observed that the computation of the radiation integral in two places (in parts 2 and 3) had code of the form:

```
CEJK = CDEXP(-AJ*AKR),
```

where `AJ = (0.d0,1.d0)`. This can be changed to:

```
CEJK = DCMPLX(DCOS(AKR),-DSIN(AKR)).
```

On the T3D, these two changes led to improved results, as can be seen in Tables 5 and 6. The run-time was reduced by 35 to 40%. When these changes were applied to the Beowulf code using the second compiler, no significant performance change was observed, leading to the conclusion that one of the optimizations performed by this compiler was similar to this hand-optimization.

Number of Processors	Original			Optimized		
	I	II	III	I	II	III
1	14.5	399	83.9	14.5	249	56.4
4	8.88	100	21.3	8.94	62.5	14.7
16	8.14	26.1	6.26	8.79	16.6	4.13

Table 5. The effect of a FORTRAN optimization, shown by T3D timing results (in seconds) for PO code, for M=40,000, N=400.

Number of Processors	Original			Optimized		
	I	II	III	I	II	III
1	0.334	85.8	1.90	0.254	48.7	0.941
4	0.147	19.8	0.354	0.149	12.2	0.240
16	0.136	4.99	0.105	0.138	3.09	0.0749

Table 6. The effect of a FORTRAN optimization, shown by T3D timing results (in minutes) for PO code, for M=40,000, N=4,900.

A comparison of the optimized Beowulf results from Tables 3 and 4 with the optimized T3D results from Tables 5 and 6 shows that when both most machines are used for this code as well as possible, the Beowulf system runs slightly faster than the T3D, but by just a few percent. This should not be taken as a general statement that the Beowulf system is equivalent in performance to the T3D, but rather as a measure of performance of running this code, specifically.

4. Conclusions

A note to the reviewers: *I do not expect the conclusions in this section to change dramatically due to increasing the computer size to 64 processors, as discussed at the beginning of section 3, though there will probably be some minor changes.*

This paper has shown that for parallel calculation of the radiation integral, a Beowulf-class computer provides slightly better performance that a Cray T3D, at a much lower cost. The limited amount of communication in the code defines it as being in the heart of the regime in which Beowulf-class computing is appropriate, and thus it makes a good test code for an examination of code performance and scaling, as well as an examination of compiler options and other optimizations.

The large test case which was run had 1.32 10^{11} floating point operations. This gives a rate of 46 MFLOP/s on one processor of the Beowulf, and 44 MFLOP/s on one processor of the T3D. These are both quite good (23% and 29% of peak, respectively.) The pipelining features of the DEC 21064 (150 MHz Alpha) processor (which is used in the T3D) seems to help the compiler achieve a larger fraction of peak performance, but the high peak rate of the Pentium Pro (200 MHz) processor (which is used in the Beowulf) closely counterbalances this.

The speed-ups for scaling from 1 to 4 processors with the large test case on the final Beowulf and T3D codes are 3.99 and 3.96, respectively, and from 4 to 16 processors, the speed-ups are 3.92 and 3.8. The overall speed-ups from 1 to 16 processors are 15.7 and 15.1. This shows that the sequential part of the code is fairly small, but still bigger on the T3D than on the Beowulf. The speed-ups for the small test case are worse, 3.92 and 3.71 going from 1 to 4 processors, 3.7 and 2.92 going from 4 to 16, and 14.5 and 10.8 going from 1 to 16. This shows the small test case to have a higher amount of sequential work than the larger case, which make sense since the I/O is not a function of the number of triangles. It is clear that scaling (for fixed size problems) through 16 processors is fairly decent. For the larger test case, the scaling is good, and for even larger problem sizes, scaling should be good for larger numbers of processors, since the I/O should remain roughly constant, but the number of floating point operations should grow roughly as M x N.

An interesting observation is that for Beowulf-class computing, using commodity hardware, the user also must be concerned with commodity software, including compilers. As compared with the T3D, where Cray supplies and updates the best compiler it has available, the Beowulf system has many

compilers available from various vendors, and it is not clear that any one always produces better code than the others. The various compilers also accept various extensions to FORTRAN, which may make compilation of any given code difficult or impossible without re-writing on some of it, unless of course the code was written strictly in standard FORTRAN 77 (or FORTRAN 90), which seems to be extremely uncommon.

It is also interesting to notice that the use hand-optimizations produces indeterminate results in the final run times, again depending on which compiler is used. Specific compiler optimization flags have not been discussed in this paper, but the set of flags that was used in each case were those that produced the fastest running code, and in most but not all cases, various compiler flag options produced greater variation in run times that any hand optimizations. The implication of this is that the user should try to be certain there are no gross inefficiencies in the code to be compiled, but that it is more important to choose the correct compiler and compiler flags. This is not a good situation.

Overall, this paper has validated the choice of a Beowulf-class computer for this physical optics application, and for other similar low-communication applications. It has examined performance of the parallel calculation of the radiation integral in terms of comparison with the Cray T3D, scaling, and compiler issues, and pointed out some "features" of which users of Beowulf-systems should be aware.

5. Acknowledgments

The authors would like to acknowledge helpful conversations with John Salmon and Jan Lindheim at Caltech, as well as the contribution of Bill Imbriale in developing the original code studied here.

6. References

[1] *Red Hat Linux Unleashed*, Sams Publishing, Indianapolis, Ind., 1996.

[2] Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. & Dongarra, J., *MPI: The Complete Reference*, The MIT Press, Cambridge, Mass., 1996.

[3] Giest, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V., *PVM: A Users' Guide and Tutorial for Networked and Parallel Computing*, The MIT Press, Cambridge, Mass., 1994.

[4] Imbriale, W. A. & Cwik, T., "A simple physical optics algorithm perfect for parallel computing architecture," In *10th Annual Review of Progress in Appl. Comp. Electromag.*, pp. 434-441, Monterey, Cal., 1994.

[5] Imbriale, W. A. & Hodges, R., "Linear phase approximation in the triangular facet near-field physical optics computer program," *Appl. Comp. Electromag. Soc. J.*, v. 6, 1991.