ASCI '97

(49)

Performance Measurements of a Light Scattering Code on the Parsytec CC

Comparison with the Parsytec PowerXplorer

Benno Overeinder and Alfons Hoekstra

Parallel Scientific Computing & Simulation Group, Faculty of Mathematics, Computer Science, Physics, and Astronomy, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, the Netherlands, tel 020 5257463, fax 020 5257490, email {bjo, alfons}@wins.uva.nl, http://www.wins.uva.nl/research/pscs/

Abstract

The performance of a Light Scattering code, which is routinely in use as a production code for more than two years now, has been measured on the Parsytec CC, a new parallel computer which has recently become available. The total execution time and communication time are measured, and compared with measurements made on an older computer (the Parsytec PowerXplorer). A simplified time complexity analysis allows to interpret the measured data in terms of three parameters and allows to understand the scalability of the code. The analysis is of value for other applications, containing comparable kernels with global communications. The data presented here will give a first impression of attainable performance on the Parsytec CC for such applications.

Keywords

performance measurements, time complexity analysis, scalability, elastic light scattering

1. INTRODUCTION

The fast Discrete Dipole Approximation (DDA) is an efficient algorithm to simulate elastic light scattering from arbitrary shaped particles. In the fast DDA a system of equations, stemming from a formal solution of the Maxwell equations in the frequency domain, is solved iteratively using a Conjugate Gradient method. The matrix-vector products in the Conjugate Gradient method can be identified as discrete convolution and, using three dimensional fft's, can be executed in an $O(N\log N)$ complexity, with N the number of unknowns (dipoles) in the method. For our purposes N needs to be very large, in the order of 10^6 or more. [1] The method itself is extensively described in Ref. [2,3], and the parallel implementation is described in Ref. [1,4,5].

Recently, in October 1996, a new parallel computer has become available for the Dutch research community. This system, a Parsytec CC, installed in Amsterdam, is a powerful distributed memory system based on PowerPC-604 processors.

In order to provide the ASCI community with an impression of the capabilities of the Parsytec CC a first set of performance measurements of the kernel of the fast DDA are shown and compared with measurements on the Parsytec PowerXplorer. More specifically, the execution time of one iteration of the parallel conjugate gradient solver of the fast DDA, using three dimensional fft's for the matrix vector products, will be shown as a function of the number of dipoles N and the number of processors p.

2. MATERIALS AND METHODS

The parallel fast DDA was originally developed in PVM on a cluster of workstations and subsequently ported to PowerPVM on the Parsytec PowerXplorer in Amsterdam. The implementation of the fast DDA in PowerPVM, running on the PowerXplorer, has been in use as a production code for approximately two year.

The main bottleneck of the parallel code is a transpose operation of the data box, resulting in a very expensive global communication operation, which is implemented in a straightforward way [4]. Each processor sets up a communication link with all other processors and exchanges the required data. Obviously, this operation can be improved. The transpose operation results in a relative bad scaling behaviour, making the code communication bound. This does not really concern us, because the main reason to use the parallel system was the availability of large amounts

of memory as compared to a single workstation. The increased computational power is also important, but the relative poor efficiency is not critical [4]. Anyway, the global communication routine is a suitable case to compare the communication capabilities of the CC with those of the PowerXplorer.

The production code was put on the Parsytec CC, and recompiled using a beta version of PowerPVM for the NanoKernel. After recompilation, which did not require any code adaptations, the program executed without errors on the Parsytec CC.

The execution time of many subroutines of the fast DDA is measured. The time required for communication is measured as well. Here, we only present the measurements made on one iteration of the fast DDA. In a typical production run we need to perform many iterations, and therefore most of the execution time will be spent in the iterative solver. Therefore, the performance and scalability of a total fast DDA run will be comparable to that of one iteration.

The Parsytec PowerXplorer contains 32 PowerPC-601 processors connected in a 4×8 grid topology. Each node has 32 Mbyte RAM. The point-to-point communication is represented by the setup time and time to send 1 byte from one processor to the next. For the PowerXplorer we measured, using PowerPVM, a setup time of 220 μs and a send time of 0.95 μs/byte [6].

The Parsytec CC contains 40 PowerPC-604 processors, of which 8 are I/O nodes containing 256 Mbyte RAM and running the AIX operating system. The other 32 processors are compute nodes, each with 96 Mbyte RAM. The processors are connected by a so-called mesh of clos (for a description of such network, see e.g. [7]). Preliminary point-to-point communication measurements show that the setup time for PowerPVM is 160 μs, and the send time is 0.1 μs/byte.

3. RESULTS

The performance of one iteration of the fast DDA is measured, as a function of the number of processors p and the number of dipoles N (i.e. the problem size). The number of processors was in the range 1 to 32; the number of dipoles was in the range $8^3 = 512$ to $160^3 = 4.096.000$. The maximum problem size on the PowerXplorer was 128^3 dipoles. The CC has enough memory to increase the problem size to $192^3 = 7.077.888$. Here, only the results for $N = 8^3$, 32^3, 128^3, and 160^3 are presented.

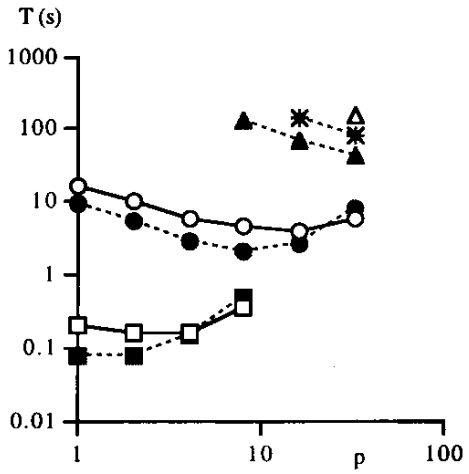

Figure 1: The execution time of 1 iteration of the fast DDA as a function of the number of processors (note the log-log axes) for a number of problem sizes N, both measured on the Parsytec PowerXplorer and the Parsytec CC. The open symbols, connected by a solid line is data for the PowerXplorer whereas the filled symbols connected by a dashed line is CC data. The squares are for $N = 8^3$, the circles are for $N = 32^3$, the triangles are for $N = 128^3$ and finally the stars are for $N = 160^3$. Note that largest problem size was only measured on the CC, as it did not fit in memory of the PowerXplorer.

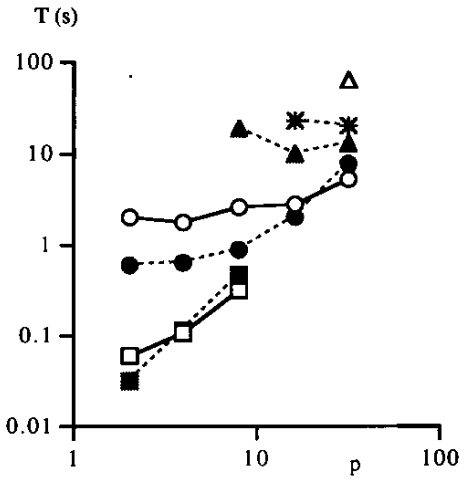

Figure 2: The total communication time during 1 iteration of the fast DDA as a function of the number of processors (note the log-log axes) for a number of problem sizes N, both measured on the Parsytec PowerXplorer and the Parsytec CC. The lines and plot symbols have the same meaning as in Fig. 1.

Fig. 1 shows the measured execution time per iteration of the fast DDA as a function of the number of processors, both on the PowerXplorer and the CC.

Fig. 2 shows the total communication time per fast DDA iteration. Because only the smallest problem sizes fit in memory of 1 processor, it is not possible to present measured speedup of efficiencies for all problem sizes. Instead the fraction of time spend in communication in one iteration of the fast DDA was calculated, and is shown in Fig. 3. This number also provides a reasonable idea of the scalability of the code, as was shown in Ref. [4].

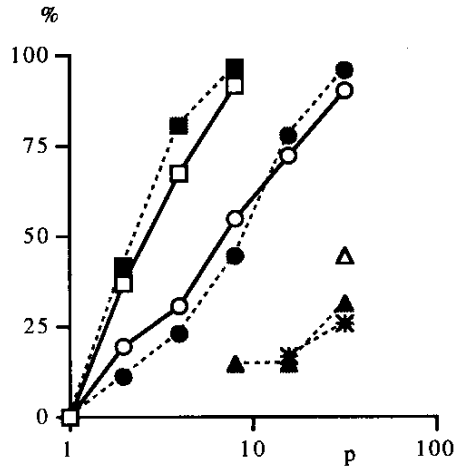

Figure 3: The fraction of communication time in 1 iteration of the fast DDA as a function of the number of processors for a number of problem sizes N, both measured on the Parsytec PowerXplorer and the Parsytec CC. The lines and plot symbols have the same meaning as in Fig. 1.

4. DISCUSSION

Global examination of Figs. 1-3 shows that the behaviour of the fast DDA code on the CC and the PowerXplorer is comparable. For each problem size one can find a minimum in the execution time as a function of the number of processors p. This minimum shifts to larger p if the problem size increases. Furthermore, this minimum lies at a smaller p for the CC, as compared to the PowerXplorer. For small N and large p the program is communication bounded, but as the problem size increases the percentage of communication decreases, resulting in increasingly better performance on larger problem instances.

A simple time complexity analysis can reproduce these effects. Define a parameter τ_{calc} which represents the total amount of computational time needed for 1 dipole. In that case the total computing time for N dipoles is

$$T_{comp}(N,p) = \frac{N}{p}(\log N)\tau_{calc}. \qquad [1]$$

The total communication time is represented by τ_{setup} and τ_{send} which respectively represent the time to initialise a point-to-point communication and the time to actually send all the data for 1 dipole. In that case the total communication time is

$$T_{comm}(N,p) = (p-1)\left[\tau_{setup} + \frac{N}{p}\tau_{send}\right], \qquad [2]$$

where we assume that each processor performs p-1 blocking synchronous point-to-point communications routines. This is obviously not the case on both the PowerXplorer and the CC and the real behaviour of the communication time is more complex, depending on the exact network topology and low-level routing strategies. Still, this model helps to understand some main features of the measured data.

The total execution time per iteration is the summation of T_{comp} and T_{comm}. Next we derive an expression for the number of processors that results in a minimum execution time. First, take the derivative to p of the total execution time.

$$\frac{\partial T}{\partial p} = -\frac{N}{p^2}(\log N)\tau_{calc} + \tau_{setup} + \left(\frac{N}{p} - \frac{(p-1)N}{p^2}\right)\tau_{send} \approx -\frac{N}{p^2}(\log N)\tau_{calc} + \tau_{setup}$$

In the second step we assume the quotient $(p\text{-}1)/p = 1$, which is valid for large p. By putting the final equation equal to zero we find p_m, the number of processors for which the execution time will be minimal,

$$p_m = \sqrt{\frac{\tau_{calc}}{\tau_{setup}} N \log N}\ . \qquad [3]$$

According to Eq. 3, p_m will shift to larger values as N increases. This is indeed observed Fig 1. Furthermore, from Fig. 1, we see that for equal N the p_m of the CC is smaller that of the PowerXplorer. That means that

$$\left(\frac{\tau_{calc}}{\tau_{setup}}\right)_{CC} < \left(\frac{\tau_{calc}}{\tau_{setup}}\right)_{PowerXplorer}. \qquad [4]$$

Based on the measured execution time on 1 processor we conclude that, for this specific code, the PowerPC-604 is a factor of 2.5 to 4 faster than the PowerPC-601. In section 2 a setup time of 220 µs for the PowerXplorer and of 160 µs for the CC was reported. These numbers confirm Eq. [4]. All this means that although the absolute speed of the CC is faster (as can be seen from Fig. 1), the decrease of execution time stops earlier on the CC. The reason for this, to Eq. 4, is that the setup time in the network of the CC did not scale equally with the increased computational power

of the processors of the CC as compared to the PowerXplorer.

According to the model, the fraction of communication time in 1 iteration is

$$fraction = \frac{(p-1)\tau_{setup} + N\tau_{send}}{\frac{N}{p}(\log N)\tau_{calc} + (p-1)\tau_{setup} + N\tau_{send}},$$

where we again assumed that $(p\text{-}1)/p = 1$. Furthermore, assuming that N is large, which allows us to neglect the terms of the setup time, we arrive at

$$fraction = \left(1 + \frac{\log N}{p}\frac{\tau_{calc}}{\tau_{send}}\right)^{-1}. \qquad [5]$$

Again, the simple model reproduces the effects as presented in Fig. 3. As N is increased, the fraction decreases, resulting in a better efficiency of the code. Furthermore, if p is increased, the fraction increases as well. This is also observed in Fig. 3. Comparing the PowerXplorer with the CC is in this case not conclusive, although for large values of p and N (e.g. $p = 32$ and $N = 96^3$ or 128^3), the fraction for the CC is always smaller that for the PowerXplorer, suggesting that

$$\left(\frac{\tau_{calc}}{\tau_{send}}\right)_{CC} > \left(\frac{\tau_{calc}}{\tau_{send}}\right)_{PowerXplorer}. \qquad [6]$$

Again, the numbers reported in section 2 and above confirm Eq. 6. However, in this case it is not considered very conclusive because, as can be seen from Fig. 3, one can also find regions where the reverse of Eq. 3 should be valid according to the model. We immediately add that the communication measurements are not fully explained by our simple model. Some main features which are predicted by Eq. [2] can be observed in the measurements shown in Fig. 2. For instance, the increasing communication time with increasing N is predicted by the model. However, detailed behaviour when increasing the number of processors is not accounted for in the model. It is obvious that the global transpose operation, which accounts for most of the communication time, behaves very differently on the CC, as compared to the PowerXplorer (e.g. the behaviour on the CC for larger problem sizes, where even a decrease of communication time is observed with increasing p). The exact reason for the behaviour on the CC is not yet understood.

On this specific application the CC is roughly 2 to 4 times faster than the PowerXplorer, depending on the problem size and number of processors. However, for small problems and a large number of dipoles the CC turns out to be slower than the PowerXplorer. As can be seen from Fig. 3, in this situation (e.g. $N = 32^3$ and $p = 32$) the total execution time is fully determined by communication, and Fig. 2 shows that the communication time on the CC is larger than on the PowerXplorer. Although this case is not relevant with respect to planned production runs using the fast DDA, it does present an interesting case to understand the network performance of the CC, as compared to the PowerXplorer, under heavy loading conditions. In the future we plan to further investigate this issue.

Finally, as we already noted in Ref. 4, good scalability of this production code is not the most important item. We are interested to run as large as possible models in a reasonable amount of time. Because of the increased amount of memory in the CC, we are now able to run models with 7.1×10^6 dipoles, as compared to models up to 2.1×10^6 dipoles on the PowerXplorer. The execution time for 1 iteration of the largest possible model in the CC is 157 seconds, which is comparable to the 149 seconds per iteration for $N = 2.1 \times 10^6$ on the PowerXplorer. In other words, we are able to run larger problems in the same amount of time, i.e. large scale computing in the spirit of Gustafson's fixed-time speedup law.

6. REFERENCES

1. A.G. Hoekstra and P.M.A. Sloot, "Coupled Dipole Simulations of Elastic Light Scattering on Parallel Systems", Int. J. Mod. Phys. C, vol 6, 663-679 (1995).
2. A.G. Hoekstra, Computer Simulations of Elastic Light Scattering, Implementation and Applications, Ph.D. thesis, University of Amsterdam, the Netherlands, 1994.
3. A.G. Hoekstra and P.M.A. Sloot, "The Discrete Dipole Approximation, Possibilities and Problems to Simulate Elastic Light Scattering", in Electromagnetic and Light Scattering - Theory and Applications, Eds. Th. Wriedt, M. Quinten, and K. Bauckhage, University of Bremen, isbn 3-88722-359-4, 1996.
4. A.G. Hoekstra, M.D. Grimminck, and P.M.A. Sloot, "Simulating Light Scattering from Micron-Sized Particles: A Parallel Fast Discrete Dipole Approximation", in High Performance Computing and Networking, Lecture Notes in Computer Science 1067, Eds. H. Lidell, A. Colbrook, B. Hertzberger, and P.M.A. Sloot, pp 269-275, 1996.
5. M.D. Grimminck, Computer Simulation of light scattering of small particles in focused laser beams, M.Sc.. thesis, University of Amsterdam, faculty of Mathematics, Computer Science, Physics, and Astrophysics, 1996.
6. A.G. Hoekstra, P.M.A. Sloot, F. van der Linden, M. van Muiswinkel, J.J.J. Vesseur, and L.O. Hertzberger, "Native and Generic Parallel Programming Environments on a transputer and PowerPC platform", Concurrency: Practice and Experience, vol. 8, pp. 19-46 (1996).
7. A. D. Pimentel and L. O. Hertzberger, "Evaluation of a Mesh of Clos Wormhole Network" in Proc. of the 3rd International Conference on High Performance Computing, IEEE Computer Society Press, Dec. 1996.