Performance Characteristics of Fast Elliptic Solvers on Parallel Platforms

Roldan Pozo *
Karin Remington †
Computing and Applied Mathematics Laboratory
National Institute of Standards and Technology ‡

Abstract

We present performance analyses of a fast three-dimensional elliptic orthogonal spline collocation (OSC) method on the Connection Machine CM-5, and various network clusters using the Parallel Virtual Machine (PVM) software package. While the OSC method is an efficient approach for solving Poisson, Helmholtz, and other separable PDE problems, it poses serious challenges for parallel architectures because of its high communication to computation ratio: $O(N^3)$ data movement to $O(N^3 \log N)$ computation. Furthermore, the communication pattern is one of the most expensive: personalized all-to-all. Results show that communication time occupies roughly 5% of total execution time in native message-passing systems (such as NX and CMMD), 10% with PVM layer on MPP's, and roughly 50% for Ethernet networks.

1 Introduction

With the advent of PVM 3, it has been possible to run PVM on conventional parallel architectures such as TMC's Connection Machine 5 and Intel's iPSC/860. This naturally brings up the question of how efficient PVM is compared to the native message passing of each architecture. Rather than study this with isolated kernels, we analyze these performance characteristics in the context of an actual application framework.

Our focus is on the Helmholtz equation, or Poisson's equation as a special case, whose solution occurs frequently in large computational problems. Many fast solvers, including the orthogonal spline collocation (OSC) method described here, require personalized all-to-all communication patterns, typically in the form of multidimensional array transpositions. These communication requirements are among the most demanding in terms of bandwidth and communication latency; they also serve to identify bottlenecks in a physical network or message passing layer.

We begin with a brief overview of OSC and the parallel implementation of matrix decomposition methods. We then compare the performance characteristics of an OSC implementation, 3DMD, using PVM 3 and native message passing kernels.

*Formerly at the University of Tennessee.

†Formerly at Oak Ridge National Laboratory.

‡Contribution of the National Institute of Standards and Technology (NIST), not subject to copyright. Certain commercial products are identified here in order to document our computational experiments. Identification of such products does not imply recommendation or endorsement by NIST, nor does it imply that the products identified are necessarily the best available for the purpose.

2 OSC and Matrix Decomposition

We focus on three-dimensional elliptic partial differential equations of the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + L_z u = f(x, y, z), \quad (x, y, z) \in \Omega = (a_x, b_x) \times (a_y, b_y) \times (a_z, b_z), \tag{1}$$

where

$$L_z u = p(z)\frac{\partial^2 u}{\partial z^2} + q(z)\frac{\partial u}{\partial z} + r(z)u,$$

and in each direction either Dirichlet, Neumann, or periodic boundary conditions are specified. To apply the method of orthogonal spline collocation (OSC), we determine an approximate solution in the space of piecewise tri-cubic Hermite polynomials by requiring that the approximation satisfy the governing PDE exactly at $(2)^3$ suitably chosen "collocation" points in each cell of the uniformly partitioned domain. Starting with a three-dimensional domain of size $N_x \times N_y \times N_z$, this requirement results in a linear system of $8N_xN_yN_z$ equations which can be written compactly in the tensor product form [4]:

$$(A_x \otimes B_y \otimes B_z + B_x \otimes A_y \otimes B_z + B_x \otimes B_y \otimes A_z)\vec{u} = \vec{f}. \tag{2}$$

Matrix decomposition (MD) can be viewed as a generic algorithm for solving systems of equations of this general form. (Such systems arise not only in collocation, but also in finite difference and finite element methods; see, for example, [5, 6, 7, 8, 9].) In the two dimensional case, the MD technique transforms the corresponding approximation problem with respect to one variable, based on eigensystem information particular to the discretization method used, thus reducing the problem to a collection of independent one-dimensional (1-D) subproblems. Once these subproblems are solved, the solution of the original approximation problem is obtained by applying the inverse of the transformation first applied. This can be extended to three dimensions by transforming with respect to two of the variables, again reducing the problems to a set of 1-D subproblems. The efficiency of the OSC algorithm we examine here comes from the ability to provide explicit formulas for the eigensystems corresponding to the one dimensional Hermite cubic collocation problems and the ability to carry out forward and inverse transformations using fast Fourier transforms (FFTs). These transformation steps are the most time consuming portion of the calculation, requiring $O(N_xN_yN_z(\log N_x + \log N_y))$ operations, while the solution of the independent 1-D problems requires only $O(N_xN_yN_z)$ operations. For a more in depth description of OSC and the related MD algorithm, see [2, 3, 4].

3 Distributed memory implementation

The highly parallel nature of the OSC-MD algorithm is evident if the data vector is viewed as a three-dimensional array throughout the computations. Each step of the algorithm can then be carried out as independent operations on a collection of subvectors of this array. For example, the system after transformation can be viewed as $4N_xN_y$ independent linear systems of order $2N_z$, where the right hand sides are subvectors of the three-dimensional array which run in the direction corresponding to the space variable z. We have chosen to parallelize the algorithm across just one of the dimensions to reduce the amount of global communication required. With this approach, parallelizing in the z-direction, each processor holds $2N_z/nprocs$ "slices" of the data array, with complete subvectors local in both the x and y directions. Therefore, in both the forward and inverse transformation phases of the MD algorithm, the subvectors of interest are completely local to each processor. However, after the forward transformation, the subvectors required for the right hand sides of the decomposed system are allocated across processors. If the data array is transposed across two dimensions, a task requiring global ("all-to-all personalized") communication, each subvector will then lie completely within a single processor. After these independent systems are solved, the data is again transposed so that the layout is as it was originally, allowing the inverse transformation to proceed without interprocessor communication.

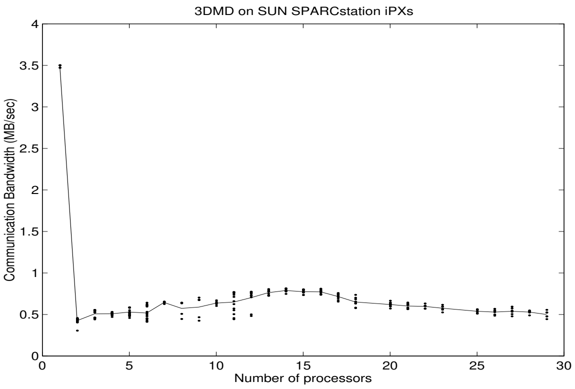

Figure 1: SPARC network performance for increasing problem sizes

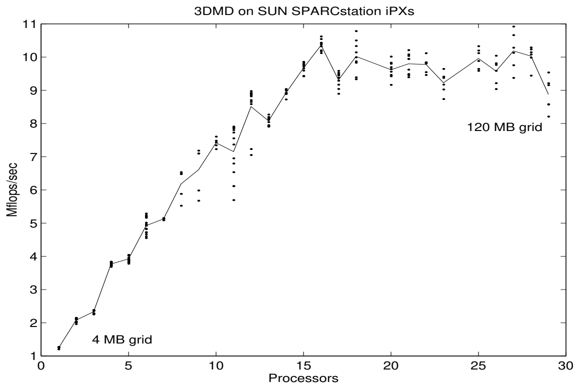

Figure 2: SPARC scaled speedup for problems with running time proportional to the number of processors

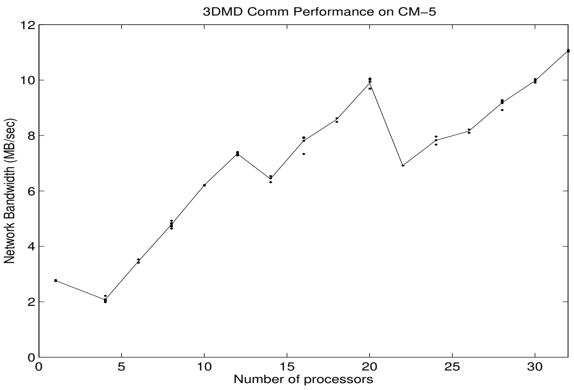

Figure 3: CM-5 network performance for increasing problem sizes

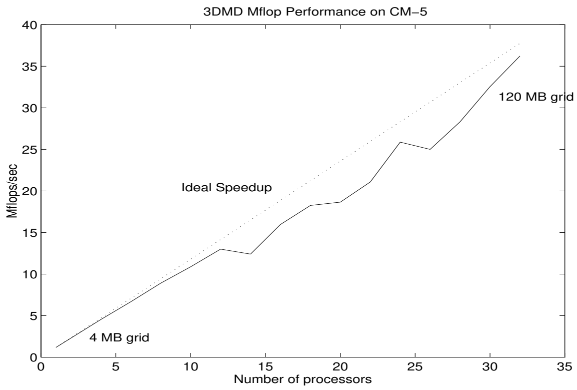

Figure 4: CM-5 scaled speedup for problems with running time proportional to the number of processors

4 Performance results

In this section we present results of timing measurements for the parallel platform implementations of 3DMD under PVM 3.3 and CMMD v. 3.2[10]. Baseline network performance studies were made on an Ethernet cluster of 32 SUN 40 MHz/ 8 MB SPARCStation IPXs at the University of Tennessee. Results in Figure 1 show that the network traffic typically saturates around 800 Kb/s, but sustains 500 Kb/s performance even for large ($p \approx 30$) configurations. It should be noted that in the single-processor case, no PVM communication calls are made; the transpose is performed at the memory-to-memory transfer rate of the IPX, about 3.3 MB/sec.

Figure 2 illustrates the speedup effects (in terms of Mflops) for problem sizes that scale with the number of processors. We chose the problem size so that each processor receives roughly 4 MBytes of data. The results show that speedups diminish after 15 processors. Fairly good efficiency (> 80%) is achieved on clusters of between 2 and 10 processors.

Figures 2 and 3 show corresponding results on a 32-processor Connection Machine CM-5 at the University of Tennessee. Although these nodes are outfitted with Weitek vector units, the native CM-5 message passing library, CMMD, does not allow access to these units. Likewise, the PVM layer (which is built atop CMMD)

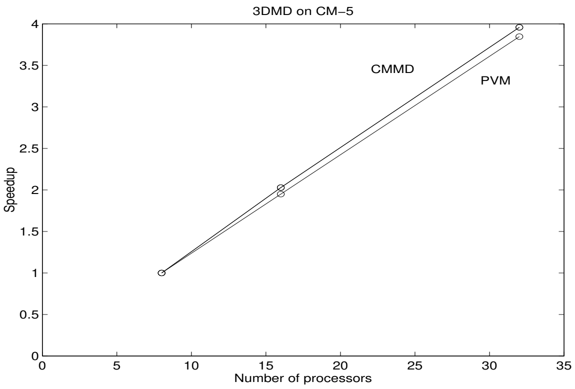

Figure 5: Comparison between native CMMD and PVM on the CM-5.

	Ether-	CM-5	
Procs	Net	CM5-PVM	CMMD
32	67%	15%	6%
16	51%	13%	5%
8	41%	11%	5%
4	30%	8%	5%

Figure 6: Communication Costs, as % of total time.

does not access these vector units. The result is that the floating point performance on each node of the CM-5 is limited by the main processing unit, a SUN SPARC chip. Nevertheless, this provides an interesting comparison with the previous workstation cluster results since the computing nodes are of a similar architecture but with a different communication network. With this setup one can study the effects of improved network bandwidth while maintaining the processing power of each node roughly constant.

Figure 3 illustrates the network bandwidth achieved on the CM-5. The single node communication performance is essentially that of the SPARC chip, about 3 MB/sec. The bandwidth falls off somewhat when $p = 4$, due to the fact that network bandwidth is slower than the memory-to-memory bandwidth. After $p = 4$ the communication bandwidth increases linearly with the number of processors. We see two discontinuities, one at $p = 14$ and $p = 21$ which may correspond to levels of the interconnect topology (fat-tree) that are being traversed.

Figure 4 illustrates the scaled speedup of the 3DMD application. It is meant to accompany the previous SUN SPARC results (Figure 2). The problem size has been scaled so that the work (e.g. number of floating operations) is roughly proportional to the number of processors. The speedup exhibited is close to optimal for $p < 12$ but still good for larger numbers of processors. At larger problems (around $p = 30$), the speedups are roughly 6-10% from optimal.

Figure 5 illustrates the speedups for a representative problem on a $64 \times 64 \times 64$ grid . The difference 3DMD between PVM and the native CMMD message passing system on the CM-5 is quite small: less than 3% for this application. Furthermore, the speedups for both interfaces are nearly linear, for 8, 16, and 32 processor configurations.

Figure 6 lists the respective communication costs (as a percentage of total execution time) for all three platforms.

5 Conclusions

We have found the OSC application to provide a useful subject for performance study on parallel platforms due to its regular and clearly delineated communication phases, together with its high communication to computation ratio. Since the communication time is spent almost entirely in the two separate 3-D data transpositions, the timing measurements for communication can be easily separated. Furthermore, with its $O(N^3)$ communication vs. $O(N^3 \log N)$ computation, it is not easy to improve performance simply by increasing problem size to a sufficient level.

The CM-5 results presented in Section 4 indicate that choosing PVM to maintain portability does not incur a significant performance degradation, even in this application with heavy communication requirements. However, as figure 6 points out, the PVM overhead increases faster (as a percentage of total execution time) than CMMD, and this could have a more visible impact on larger ($p > 32$) processor configurations.

We also note that although the time spent in communication is fairly small on MPP systems, it accounted for the majority of the elapsed time in most Ethernet/workstation configurations tested. This is in agreement with the general observation that large task granularity is necessary for reasonable performance on local Ethernet networks.

References

[1] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, V. Sunderam, *User's Guide to PVM: Parallel Virtual Machine*, ORNL/TM-11826, Mathematical Sciences Section, Oak Ridge National Laboratory, Sept. 1991.

[2] K. R. Bennett, "Fast Direct Solution of Three-Dimensional Poisson and Helmholtz Problems on Distributed Memory Machines," *Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, March 22-24, 1993.

[3] B. Bialecki and K. A. Remington, *Fourier Matrix Decomposition Methods for the Least Squares Solution of Singular Neumann and Periodic Hermite Bicubic Collocation Problems,* SIAM J. Sci. Comp., to appear.

[4] B. Bialecki, G. Fairweather, K. R. Bennett, *Fast Direct Solvers for Piecewise Hermite Bicubic Orthogonal Spline Collocation Equations,* SIAM J. Numer. Anal., 29, 1992, pp. 156-173.

[5] R. F. Boisvert, *Algorithm 651: Algorithm HFFT – High-Order Fast-Direct Solution of the Helmholtz Equation*, ACM Trans. Math. Software, 13 (1987), pp. 235–249.

[6] B.L. Buzbee, G.H. Golub, and C.W. Nielson, *On Direct Methods for Solving Poisson's Equations*, SIAM J. Numer. Anal. 7 (1970), pp. 627–656.

[7] L. Kaufman and D. Warner, *Algorithm 685: A Program for Solving Separable Elliptic Equations*, ACM Trans. Math. Software, 16 (1990), pp. 325–351.

[8] R. E. Lynch, J. R. Rice, and D. H. Thomas, *Direct Solution of Partial Difference Equations by Tensor Product Methods*, Numer. Math., 6 (1964), pp. 185–199.

[9] P.N. Swarztrauber, "Fast Poisson Solvers", in *Studies in Numerical Analysis*, (editor G.H. Golub) MAA Studies in Mathematics, Vol. 24, Mathematical Association of America, 1984, pp. 319–370.

[10] Thinking Machines Corporation, "CMMD User's Manual," Boston, Mass., 1993.