Multitasking Domain Decomposition Fast Poisson Solvers on the Cray Y-MP

Tony F. Chan* Rod A. Fatoohi†

Report RNR-90-005, March 1990

Abstract

We present the results of multitasking implementation of a domain decomposition fast Poisson solver on eight processors of the Cray Y-MP. The object of this research is to study the performance of domain decomposition methods on a Cray supercomputer and to analyze the performance of different multitasking techniques using highly parallel algorithms. Two implementations of multitasking are considered: macrotasking (parallelism at the subroutine level) and microtasking (parallelism at the do-loop level). A conventional FFT-based fast Poisson solver is also multitasked. The results of different implementations are compared and analyzed. A speedup of over 7.4 on the Cray Y-MP running in a dedicated environment is achieved for all cases.

Appeared in the Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing, (Chicago, December 1989), SIAM Publication, pp. 237 – 244.

*Department of Mathematics, UCLA, Los Angles, CA 90024. This work was funded in part by Contract NCC2-387 between NASA and USRA.

†NAS Applied Research Office, NASA Ames, M/S T045-1, Moffett Field, CA 94035. The author is an employee of Sterling Software. This work was funded in part by NASA Contract NAS2-11555.

1 Introduction

Domain decomposition is a class of numerical algorithms for solving partial differential equations on a spatial domain by combining (usually iteratively) solutions of smaller independent problems on (overlapping or non-overlapping) subdomains. These methods have received a lot of study recently [1, 5], partially because of their inherent large-grain parallelism, and appear to ideally fit coarse-grain architectures with a small to moderate number of powerful processors (such as the Cray Y-MP). A fundamental implementation issue is the effect of the communication and synchronization overhead on the overall performance. While a few previous authors have addressed this issue [3, 6], there is relatively little work on the new Cray machines. Our work here is a preliminary step in this direction.

We chose a simple and yet representative (in terms of global communication requirements) domain decomposition algorithm (a fast Poisson solver [2]) and study its performance on an eight-processor Cray Y-MP, implemented with both macrotasking and microtasking. In addition, a conventional FFT-based solver is also multitasked and the performance results compared and analyzed. A relative speedup of over 7.4 running in a dedicated environment is achieved for all cases.

2 The Numerical Algorithms

In this section, we briefly describe the Fast Direct Solver (FDS) algorithm, the basic domain decomposition algorithm (DD) and a more efficient version (DD1). The domain decomposition algorithms were derived in [2] and we shall skip the derivation and only present the algorithms in a schematic form sufficient for describing the macrotasking and microtasking.

All three algorithms are for solving the Poisson equation $\Delta u = f$ on a rectangular region discretized with a uniform mesh with grid size h and n interior grid points in the $x-$direction and m interior grid points in the $y-$direction. The Poisson equation is discretized by a standard five point second order difference approximation which results in a linear system $Au = f$, where $A = trid(I, T, I)$ is block tridiagonal of size mn by mn, and $T = trid(1, -4, 1)$ is n by n. For the description of the algorithms, we view u and f as 2D arrays, with the index (j, i) corresponding to the grid point $(x, y) = (i * h, j * h)$. The notation u_{j*} denotes the $n-$vector $(u_{j1}, ..., u_{jn})^T$. Quantities with " ˜ " denote transformed variables.

It is well known that T can be diagonalized by the sine transform (symmetric and orthogonal) matrix W, with $(W)_{ij} = \sqrt{2h}\sin(ij\pi h)$. Using a block diagonal transform with W, we can transform the system $Au = f$ into a set of n independent tridiagonal systems, from which we obtain the FDS.

Algorithm FDS

1. FFT in x-direction: $\tilde{f}_{j*} = W f_{j*}, \qquad j = 1 : m$
2. Tridiagonal Solves in y-direction: $\tilde{u}_{*i} = \tilde{T}(m)^{-1}\tilde{f}_{*i}, \qquad i = 1 : n$
3. FFT in y-direction: $u_{j*} = W\tilde{u}_{j*}, \qquad j = 1 : m$

Here, $\tilde{T}(m)$ is a m by m tridiagonal matrix defined by: $\tilde{T}(m) = trid(1, -2 - \sigma_i, 1), \quad i = 1 : m$, where $\sigma_i \equiv 4\sin^2(i\pi h/2)$.

For the domain decomposition algorithms, the domain is divided into p subdomains (stripes) by $p-1$ interfaces along the x-direction. We shall denote the index of the interfaces by $j_k, \quad k = 1 : p-1$. Thus $m_k \equiv j_{k+1} - j_k - 1$ is the number of grid points in the $y-$direction in the $k-$th subdomain. When all the subdomains are equal in size, we denote this number by $m_0 \equiv (m - p + 1)/p$. The domain decomposition algorithms first perform subdomain solves (we use the FDS from above) with zero boundary conditions. Then a set of n independent tridiagonal systems governing the unknowns on the interfaces are solved, after which the solutions inside the subdomains are computed by a second subdomain solve. For the coupling of the unknowns across different interfaces, we need to define the following $p-1$ by $p-1$ tridiagonal matrix: $\hat{T}_j = trid(\delta_j, \lambda_j^C, \delta_j)$, where δ_j and λ_j^C are defined in [2]. We shall use the notation $\tilde{u}_{*i}^{(k)}$ to denote the vector $(\tilde{u}_{j_{k-1}+1}, ..., \tilde{u}_{j_k-1})^T$.

Algorithm DD

1. Subdomain Solves: For $k = 1 : p$
   - 1.1 $\tilde{f}_{j*} = W f_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$
   - 1.2 $\tilde{u}_{*i}^{(k)} = \tilde{T}(m_k)^{-1} \tilde{f}_{*i}^{(k)}, \qquad i = 1 : n$
   - 1.3 $u_{j*} = W \tilde{u}_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$
2. Solve interface system:
   - 2.1 $u_{j_k*} = f_{j_k*} - u_{(j_k+1)*} - u_{(j_k-1)*}, \qquad k = 1 : p-1$
   - 2.2 $\tilde{u}_{j_k*} = W u_{j_k*}, \qquad k = 1 : p-1$
   - 2.3 $v^{(i)} = \hat{T}_i^{-1} v^{(i)}, \quad v^{(i)} \equiv (\tilde{u}_{j_1 i}, ..., \tilde{u}_{j_{p-1} i})^T, \qquad i = 1 : n$
   - 2.4 $u_{j_k*} = W \tilde{u}_{j_k*}, \qquad k = 1 : p-1$
   - 2.5 $f_{(j_k+1)*} = f_{(j_k+1)*} - u_{j_k*}, \qquad k = 1 : p-1$
   - 2.6 $f_{(j_k-1)*} = f_{(j_k-1)*} - u_{j_k*}, \qquad k = 1 : p-1$
3. Subdomain Solves: For $k = 1 : p$
   - 3.1 $\tilde{f}_{j*} = W f_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$
   - 3.2 $\tilde{u}_{*i}^{(k)} = \tilde{T}(m_k)^{-1} \tilde{f}_{*i}^{(k)}, \qquad i = 1 : n$
   - 3.3 $u_{j*} = W \tilde{u}_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$

Note that Algorithm DD requires two solves on each subdomain. However, there are certain redundancies involved. For example, the u variables in Step 2.1 are obtained from the inverse transform in Step 1.3 and are then immediately forward transformed in Step 2.2. By solving the interface system in the transformed space, we can eliminate this redundancy. Moreover, with this modification, Step 3.1 can also be eliminated because the transforms were already computed in Step 1.1. We denote this more efficient algorithm by DD1 [2]:

Algorithm DD1

1. Subdomain Solves: For $k = 1 : p$
   - 1.1 $\tilde{f}_{j*} = W f_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$
   - 1.2 $\tilde{u}_{*i}^{(k)} = \tilde{T}(m_k)^{-1} \tilde{f}_{*i}^{(k)}, \qquad i = 1 : n$

2. Solve interface system:

   2.1 $\tilde{f}_{j_k*} = W f_{j_k*}, \qquad k = 1 : p-1$

   2.2 $\tilde{u}_{j_k*} = \tilde{f}_{j_k*} - \tilde{u}_{(j_k+1)*} - \tilde{u}_{(j_k-1)*}, \qquad k = 1 : p-1$

   2.3 $v^{(i)} = \hat{T}_i^{-1} v^{(i)}, \quad v^{(i)} \equiv (\tilde{u}_{j_1 i}, ..., \tilde{u}_{j_{p-1} i})^T, \qquad i = 1 : n$

   2.4 $\tilde{f}_{(j_k+1)*} = \tilde{f}_{(j_k+1)*} - \tilde{u}_{j_k*}, \qquad k = 1 : p-1$

   2.5 $\tilde{f}_{(j_k-1)*} = \tilde{f}_{(j_k-1)*} - \tilde{u}_{j_k*}, \qquad k = 1 : p-1$

3. Subdomain Solves: For $k = 1 : p$

   3.1 $\tilde{u}_{*i}^{(k)} = \tilde{T}(m_k)^{-1} \tilde{f}_{*i}^{(k)}, \qquad i = 1 : n$

   3.2 $u_{j*} = W \tilde{u}_{j*}, \qquad j = j_{k-1} + 1 : j_k - 1$

   3.3 $u_{j_k*} = W \tilde{u}_{j_k*}, \qquad k = 1 : p-1$

3 Macrotasking and Microtasking

Macrotasking (MA) is the process of partitioning a program into two or more tasks at the subroutine level. The granularity of these tasks may be large. The system software for the CRAY Y-MP provides a library of Fortran-callable subroutines that implements a basic set of primitive macrotasking functions. These subroutines are called by a macrotasked program as required to create and synchronize task execution; see [4] for more details.

Microtasking (MI) is the process of partitioning a program into parts at the do-loop level. The granularity of these parts may be small. Microtasking is specified by a number of user-supplied directives that appear as comment lines. A preprocessor, called premult, interprets the directives and then rewrites the code to make microtasking library calls. The addition of the microtasking directives does not reduce the portability of the code.

4 Implementation and results

We implemented algorithms FDS and DD1 on the Cray Y-MP at NASA Ames Research Center. At the time of our implementation, the machine had eight processors, 32 Mwords of main memory, and 6.3 nsec clock cycle. Both algorithms were macrotasked and microtasked on the Y-MP using two, four, and eight processors.

Algorithm FDS (see Section 2) was multitasked by parallelizing each of the three steps of the algorithm separately and separating these steps by synchronization points; two synchronization points were needed. The FFT parts of the algorithm (Steps 1 and 3) were macrotasked by having each processor computes the sine transforms of a block of m/p columns while with microtasking each processor computes the sine transform of a single column and then asks for another one and so on. This is the main difference between the two implementations. Each block of n/p rows of the tridiagonal systems (part 2) were solved by a processor for both implementations.

Multitasking Algorithm DD1 (see Section 2) was quite simple, since the domain has already been partitioned. The implementation required two synchronization points: one after Step 1 and the other surrounding Step 2.3; this step was implemented sequentially on a single processor since it runs across n systems and it is

Table 1: Operation Counts

FDS	DD	DD1
m FFT(n)	pm_0 FFT(n)	pm_0 FFT(n)
n TRI(m)	pn TRI(m_0)	pn TRI(m_0)
m FFT(n)	pm_0 FFT(n)	
	$(p-1)$ FFT(n)	$(p-1)$ FFT(n)
	n TRI($p-1$)	n TRI($p-1$)
	$(p-1)$ FFT(n)	$(p-1)$ FFT(n)
	pm_0 FFT(n)	
	pn TRI(m_0)	pn TRI(m_0)
	pm_0 FFT(n)	pm_0 FFT(n)

very inexpensive. No synchronization point was required between Steps 2 and 3 since Steps 2.5 and 2.6 can be performed locally for each subdomain. There is no difference between the two implementations except that different primitives were used. The microtasked implementation of DD1 represents a case of using small granularity primitives to solve a problem with a large granularity .

Fig. 1 through 6 show the execution time (measured in a dedicated environment), speedup and efficiency of algorithms FDS and DD1 on the Cray Y-MP using both macrotasking (MA) and microtasking (MI) for two square domains, 511×511 and 1023×1023. Speedup was computed by taking the ratio of the time to solve the problem using one processor to the time to solve the same problem using p processors. The time to run Algorithm DD1 on a single processor was considered in computing the speedup for the macrotasked and microtasked versions of the algorithm. Efficiency was determined by taking the ratio of the speedup using p processors to p. The execution time measure (Fig. 1 and 2) shows that Algorithm FDS is faster than Algorithm DD1 on a single processor. This means that there is some overhead involved in the domain decomposition method. Table 1 shows operation counts for the three algorithms: FDS, DD, and DD1. If we compare DD1 with FDS, we can see that the main overhead with DD1 is the term [pn TRI(m_0)] which resulted in about 9% overhead compared to FDS. The execution time measure also shows that the performance of both macrotasking and microtasking is comparable for both algorithms. A speedup ranging between 7.43 and 7.75 and an efficiency ranging between 93% and 97% were obtained for these implementations, as shown in Fig. 3 through 6. These figures also show that the four implementations achieved good performance on the Cray Y-MP, with the microtasked FDS has a slight advantage over the other implementations for the eight processor case. A slight improvement in the performance of these implementations were noticed for the larger problem (1023×1023) because of a better computation to synchronization ratio for this problem.

5 Conclusion

Our results show that high speedup is achievable on the Cray Y-MP for the highly parallel domain decomposition algorithms that are considered. The overhead for the multitasking primitives are relatively small and are also easy to use. The results for macrotasking and microtasking are comparable, primarily because the special domain decomposition algorithms we considered here possess fine grain as well as coarse grain parallelism. The situation would favor macrotasking if a less parallelizable subdomain solver is used (as is necessary for more complicated problems). The algorithmic overhead of Algorithm DD1 over the FDS is small but noticeable, and in fact FDS is the faster solver, but the situation would also change for more complicated problems (e.g. irregular domains and more complicated elliptic operators) for which FDS cannot be directly applied.

References

[1] T. F. Chan, R. Glowinski, J. Periaux and O. B. Widlund, editors, *Domain Decomposition Methods*, SIAM, Philadelphia, 1989.

[2] T. F. Chan and D. C. Resasco, *A Domain-Decomposed Fast Poisson Solvers on a Rectangle*, SIAM J. Sci. Stat. Comp., Vol. 8, No. 1, 1987, pp. s27 – s42.

[3] T. F. Chan and D. C. Resasco, *Hypercube Implementation of Domain-Decomposed Fast Poisson Solvers*, in Hypercube Multiprocessors 87, Mike Heath, editor, SIAM, 1987.

[4] R. A. Fatoohi, *Multitasking a Navier-Stokes Algorithm on the CRAY-2*, The Journal of Supercomputing, Vol. 3, No. 2, 1989, pp. 109 – 124.

[5] R. Glowinski, G. H. Golub, G. Meurant and J. Periaux, editors, *Domain Decomposition Methods for Partial Differential Equations*, SIAM, Philadelphia, 1988.

[6] W. D. Gropp and D. E. Keyes, *Domain Decomposition on Parallel Computers*, Research Report YALEU/DCS/RR-723, Dept. of Comp. Sci., Yale Univ., 1989.