DTIC FILE COPY

(4)

AD-A206 389

DTIC
ELECTE
APR 05 1989
S D
D

Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2

Harry Berryman , Joel Saltz and William Gropp

YALEU/DCS/TR-685
March 1989

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

089

(4)

DTIC
ELECTE
APR 05 1989
S D
D CS

Yale University
Department of Computer Science

Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2

Harry Berryman , Joel Saltz and William Gropp

YALEU/DCS/TR-685
March 1989

This work has been supported in part by the U.S. Office of Naval Research under Grant N00014-86-K-0310 through Yale and by DARPA contract DAA-1101-88-C-0409 through Scientific Computing Associates

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2 *

Harry Berryman
Joel Saltz
William Gropp
Department of Computer Science
Yale University
New Haven, CT 06520

February 24, 1989

Accesio For
NTIS CRA&I
DTIC TAB
Unanno
Justificatio
By per NP
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

1 Abstract

In the work presented here, we measured what might be regarded best case timings for the sparse matrix vector multiplies, sparse triangular solves, and inner products that constitute the iterative portion of Krylov space programs that use incompletely factored matrices for preconditioning. We performed timings on a large three dimensional model problem over a cube shaped domain discretized with a seven point template. The highest computational rate we were able to achieve for the sparse triangular solve was 13.1 MFlops on 4K processors. This would correspond to 210 MFlops on an appropriately scaled problem on a 64K processor machine. The highest computational speed we achieved for a matrix vector multiply was 64.2 MFlops. This would correspond to a speed of 1027.0 MFlops in a 64K processor machine. Thus for appropriately structured problems, the CM-2 achieves impressive computational speeds. We compared the computational speed obtained from the CM-2 with those from optimized code on a single head of a Cray X/MP. For both the matrix vector multiply and triangular solve computational kernels, the projected speeds on a full 64K processor machine exceed the Cray X/MP speeds by a substantial factor.

The timings on the CM-2 were highly dependent on the use of very specialized pro-

*This work was supported by the U.S. Office of Naval Research under Grant N00014-86-K-0310 through Yale and by DARPA contract DAA-1101-88-C-0409 through Scientific Computing Associates

grams. These programs mapped the problem domain onto the processor topology very carefully and used the optimized local NEWS communications network. We explored the consequences of relaxing these restrictions in a variety of ways. Performance degraded very significantly as we abandoned the optimized NEWS communications network and abandoned the careful mapping of a problem domain to the CM-2 processor topology. In some cases we noted 20 to 75 fold degradations in performance as these constraints were relaxed.

2 Overview

There are at least two critical components required to obtain extremely fast methods for solving linear systems. One is the use of efficient and robust numerical algorithms, and the other is the employment of effective techniques for delivering a large amount of computing power. These requirements can conflict with one another in a variety of ways. Many modern methods of solving reasonably general classes of linear systems involve a degree of implicitness; this implicitness can limit the amount of available concurrency.

When Krylov space linear solvers are used, the choice of preconditioner can play a major role in determining the operation count of the resulting algorithm [12], [9], [8], [11]. Unfortunately, some of the most powerful preconditioners are obtained by using incompletely factored matrices. To apply these preconditioners, we must repeatedly solve sparse triangular systems. The efficiency with which such solutions could be carried out was characterized by Saad et. al. [13]. Sparse triangular systems arising from a range of problems have been efficiently solved on a number of shared memory architectures [10], [1], [3], [4]. Because data dependencies limit the concurrency available from a sparse triangular solve, it has not been clear that triangular solves could be usefully employed in programs written for massively parallel architectures. One goal of our study was to determine whether one could realistically hope to take advantage of incompletely factored matrices as preconditioners for solving sparse linear systems on massively parallel architectures such as the CM-2.

Several other researchers have addressed relative utility of various forms of preconditioning on the CM-2. Both [6] and [7] concluded that it was not worthwhile to precondition using incompletely factored matrices when solving linear systems arising from two dimensional partial differential equations on massively parallel machines such as the CM-2. We concentrated our efforts on a simple three dimensional model problem because of the greater parallelism found in the sparse triangular solve.

In the work described in this paper, we have carefully examined a set of model problems and have been able to demonstrate that under the appropriate circumstances we are able to obtain surprisingly good performance on key computational kernels including the sparse triangular solves used for preconditioning in Krylov space solvers. To put our results in perspective, we compared our measured and projected computational

speeds with results from experiments on a single head of a Cray X/MP. We concluded that for selected problems, a 64K processor CM-2 could outperform an analogous well tuned program single head of a Cray X/MP by at least a factor of 5 in performing the sparse triangular solves and the matrix vector multiplies required in the Krylov solver inner loop.

Efficient methods for solving partial differential equations frequently make use of non-uniform grids designed to put the most computational effort where the problem is hardest. An effect of this approach is that the algebraic linear (and non-linear) systems that must eventually be solved are sparse and quite irregular in structure. Careful mapping of workload can be extremely important in obtaining adequate performance from multiprocessor architectures with strong memory hierarchies; mapping is typically straightforward in very regular problems with a known structure and is much more problematic for problems with unknown or irregular structures.

Another goal of our research was to quantify the degree to which performance on a machine such as the CM-2 depends on exploiting regularities in problem structure. We will address the issues raised above by presenting a number of timings on the CM-2 for sparse matrix vector multiplies, sparse triangular solves and inner products. We find that obtaining good performance on the CM-2 is critically dependent on the use of very well tuned special purpose computational kernels. We have described a number of experimental results arising from our investigations in [5].

In Section 3, we will present what we might regard as best case timings for the sparse matrix vector multiplies, sparse triangular solves, inner products and SAXPYs that constitute the iterative portion of Krylov based programs. It will be shown that for large three-dimensional problems, good performance can be obtained from sparse triangular solves. In Section 4, we demonstrate that performance on the CM-2 is an extremely sensitive function of 1) problem mapping and 2) *a priori* knowledge of dependency patterns. We will show that this performance sensitivity shows up very strongly not only in the sparse triangular solve but also in the sparse matrix vector multiply.

3 Performance on a Regular Three Dimensional Mesh

In this section we describe the results of experiments that give a best case estimate of the rate with which the CM-2 can carry out the procedures that make up the iterative portions of Krylov space linear equation solvers. We will first present timings from consecutive sweeps over a three dimensional mesh along with timings for the corresponding inner products and SAXPYs. The performance measurements we present here characterize the performance that would arise from the iterative portions of linear solvers

employing many simple preconditioners. We will then present timing results from a program that performs a sequence of linked matrix vector multiplies and triangular solves. We argue that the matrix vector multiply and triangular solve timings obtained from this benchmark are a fair measure of the timings that would be observed from these procedures were they integrated into an iterative loop of the appropriately preconditioned Krylov solver. As part of this test loop, we also measured the time required to perform inner products in a manner that conformed with data structures and the mapping used for the other two procedures.

The software provided with the CM-2 makes use of the concept of *virtual processors*; one can program the CM-2 so that it appears that there are a larger number of processors than actually exist. The CM-2 software assigns blocks of virtual processors to each real processor. This assignment of multiple virtual processors to each real processor tends to amortize the overhead of transmitting each instruction to the physical processors. In most of the problems we investigate in this paper, increasing the ratio of virtual to real processors also reduces overheads due to communication.

3.1 Mesh Sweeps

One can use a very large number of virtual processors in implementing a sparse matrix vector multiply. We examined the performance of a very specialized PARIS program (PARIS is the CM-2 assembly language) which was written for a three dimensional problem on a cube with a seven point operator. The program hence consisted of a sequence of sweeps over a three dimensional mesh, the mesh was embedded into a cube of virtual processors with one mesh point assigned to a virtual processor. The cube of virtual processors used by the program has an edge size equal to a power of two. Subject to this constraint, the largest mesh we could embed was one with an edge size of 64. Each iteration of the 1000 carried out took an average of 49 milliseconds. This corresponds to a speed of 64.2 MFlops on the 4K processors. With an appropriately scaled problem, one might expect to obtain 1027.0 MFlops on a 64K processor machine. The results obtained from timings for meshes of varying sizes on 4K processors are depicted in Table 1.

In Table 1 we also depict measurements obtained from SAXPYs and inner products over three dimensional domains. All of these results were obtained by timing 100000 consecutive iterations. Because SAXPYs do not require communication, we expect to obtain extremely high performance. For SAXPYs carried out over a cube of virtual processors with edge size 128, we obtained a speed on 4K processors of 235.0 MFlops, this would correspond to 3760 MFlops on a 64K processor machine. The efficiency with which SAXPYs are performed decreases when one uses fewer virtual processors. A cube with edge size 16 has one virtual processor for each actual processor, from Table 1 we see that the speed of this computation decreases to 131.0 MFlops, note that this reduction in speed must be unrelated to communication overhead. In Table 1, we also

Table 1: Three Dimensional Embedding : Mesh Sweeps, Inner Products, SAXPYs : 4K processors

Grid Size edge	MVM MFlops	SAXPY MFlops	Inner Product MFlops
16	23.5	131.0	14.6
32	46.1	226.0	81.3
64	64.2	233.9	185.8
128	N.A.	235.0	220.9

```
do 100 times

   Mx = y
   Solve Lx = y
   z = inner-product(y,y)

 end do
```

Figure 1: Sweeps over a Mesh

present timings for inner products carried out over a cube of virtual processors with varying edge size.

3.2 Performance from Iterative Loops with Triangular Solves

We next present timing results from a program that performs a sequence of linked matrix vector multiplies and triangular solves. Let `M` represent a matrix obtained from the uniform discretization of a cube with a seven point template and let `L` represent a lower triangular matrix with the same sparsity structure as `M`. We carried out the test calculation depicted in Figure 1.

This program carried out the matrix vector multiply $Mx = y$ by sweeping over a three dimensional mesh. We embedded the three dimensional mesh into a two dimensional grey coded processor lattice. The sparse lower triangular system of equations $Lz = y$ was then solved by sweeping over another three dimensional mesh, embedded in a conforming fashion into the same two dimensional processor lattice. The sweep used to solve the sparse triangular system was carried out in a manner that respected the dependencies of the problem. As part of the test loop, we also measured the time required to perform inner products in a manner that conformed with data structures

Table 2: Matrix Vector Multiply, Triangular Solve, Inner Product: Optimized: 4K processors

Grid Edge Size	Inner Product (ms)	Matrix Vector (ms)	Triangular Solve (ms)	Total Time (ms)
16	1.5	29.0	51.9	81.3
32	2.8	56.7	106.8	162.4
48	3.8	78.4	169.1	240.6
64	5.0	115.9	225.7	321.5
128	22.2	478.3	960.1	1511.5

and the mapping used for the other two procedures.

We now describe in more detail how the triangular solve was carried out. In a cube with n points along any edge, i, j, k between 1 and n are used to define the position of a point in the cube where i, j, k represent the cartesian coordinates of a point in the 3-D mesh. We can parallelize this three dimensional triangular solve by concurrently solving, for each consecutive v, the plane of points satisfying the condition $i + j + k = v$ for $1 \leq v \leq 3n - 2$. Each processor in the lattice contained, for a given i and j, variables corresponding to values of k between 1 and n.

For 4K processors Table 2 depicts the timings obtained for various size domains, timings from 100 iterations were averaged. The timings obtained from the cube with edge sizes 128 and 64 corresponded to 188.9 and 104.9 MFlops respectively for the inner product, 52.6 and 27.1 MFlops respectively for the matrix vector product and 13.1 and 7.0 MFlops respectively for the triangular solve. From the above results we conclude that for an appropriately scaled problem on 64K processors, the fastest results we could obtain for the inner product, matrix vector multiply and triangular solve would be 3022 MFlops, 842 MFlops and 210 MFlops respectively.

We were able to prevent the need to move data when we followed the the matrix vector multiply by a sparse triangular solve because of the way in which we assigned data to processors. This method of data assignment did have the side effect of requiring us to perform the sparse matrix vector multiply in n consecutive phases. As one might expect, the use of this embedding does can be seen to exact a performance penalty when compared to the embedding discussed in Section 3.1. For example we attained a computational speed of 64.2 MFlops for the problem with edge size 64 in Section 3.1, we attain a computational speed of only 27.1 MFlops with the embedding discussed in this section. The conforming inner product also had to be performed in n consecutive phases.

3.3 Comparison with Cray Timings

To put the CM-2 timings presented above into perspective we will present timings from a single head of a Cray X/MP for triangular solves and matrix vector multiplies arising from problems with the same structure as the problem we presented above. We used PCGPAK, a commercially available Krylov space solver. In this program, the computation of the matrix vector multiply and triangular solve in the iterative loop were vectorized using Cray Assembly Language (CAL). Computational speeds of 24 MFlops and 22 MFlops were obtained when performing matrix vector multiplies that arose from meshes with edge sizes 20 and 30 respectively. Note that the speed of the Cray did not increase with increasing problem size, on a vector machine such as the Cray X/MP, the computational rate depends chiefly on the problems structure rather than its size. By using a more specialized data structure that was well suited for vectorization, [2] reports speeds of 150 MFlops on a single head of an Cray X/MP. The speed of the matrix vector multiply in largest problem described in Section 3.2 was 52.6 MFlops for the 4K processor machine, this should increase to 842 MFlops in a 64K processor machine on an appropriately scaled problem.

The triangular solve comparisons were also very favorable. For meshes with edge sizes of 20 and 30 a single head of an X/MP achieved computational rates of 13 and 12 MFlops respectively using PCGPAK. Ashcraft reports speeds of 25-40 MFlops on triangular solves in problems with the same structure, again his higher computational rates were achieved through the use of specialized data structures. The computational speed achieved by the largest problem described in Section 3.2 was 13.1 MFlops, this should increase to 210 MFlops in a full 64K machine for an appropriately scaled problem.

3.4 Summary

The benchmarks described in this section imply that one can achieve extremely respectable performance when one is able to make use of a highly tuned, special purpose PARIS program for carrying out the iterative portion of a large three dimensional computation. Our experiences with the higher level languages *lisp and C* were less satisfactory. For instance, our *lisp versions of the sparse triangular solve described above required much more time to run than did the PARIS version.

4 The Importance of Careful Embedding

In section 3 we described how it was possible to achieve very respectable rates of computation even on iterative loops that included a sparse triangular solve. In this section, we present some benchmarks that yield insight on what is required for achieving this high performance.

Table 3: Matrix Vector Multiply: Explicitly mapped News Net v.s. General Router: 4K processors

Grid Size	News Total (ms)	News Comp (ms)	General Total (ms)	General Comp. (ms)
64 x 64	2.25	0.93	45.56	0.99
128 x 128	4.62	1.88	189.44	1.89
256 x 256	13.11	5.46	1001.11	5.51

4.1 The Consequences of a Poor Mapping

We have found that 20 to 75 fold performance differences are observed when we compared the performance of two versions of a sparse matrix vector multiply program (Table 3). The problem consisted of sweeps over sparse matrices generated by square domains of varying sizes with five point templates. The first version is explicitly mapped onto the machine in a way that allows us to utilize the CM-2's fast NEWS network for local communication. The other version uses a general router designed to carry out arbitrary patterns of interprocessor communication. Both versions were programmed in *Lisp and used the same data structures to represent the sparse matrices. For the 256 by 256 problem the timings for the explicitly mapped NEWS network code corresponded to a speed of 40.0 MFlops, the timings for the router version of the code achieved a speed of 0.5 MFlops. Note that the cost of computation does not vary significantly with the mapping and choice of communications method; this consistency provides a reassurance that the timing differences noted are actually due to communication related costs. The timings depicted here are averages obtained from 1000 iterations of the matrix vector multiply code. From this benchmark, it is quite clear that satisfactory performance cannot be obtained even from the most rudimentary sparse matrix code if one were to map sparse matrices onto the CM-2 without regard to data dependency patterns.

4.2 The Consequences of a Using the General Router in a Well Mapped Problem

We next attempt to obtain some rough estimates of the performance we could expect if we were to write an iterative loop of a Krylov linear solver in a program capable of mapping reasonably general sparse matrices onto the CM-2. We must assume that the sparse matrix could arise from a mesh with irregularities. We consequently must use the general router rather than the NEWS network.

We will present benchmarks that attempt to quantify the effects of using the general router on a well mapped problem. This should give us a best case estimate of the per-

Table 4: Three Dimensional Mesh Sweep and Solve: News Net v.s. Send and Fetch General Router: 4K processors

Function	NEWS (ms)	Router Send (ms)	Router Fetch (ms)
Matrix Vector	116	249	876
Solve	226	374	1376
Total	342	606	2247

formance we might expect from using the general router. We examined the performance of versions of the PARIS program described in Section 3.2 in which we performed data *fetches* or data *sends* using the general router instead of performing data sends using the NEWS network. Note that in all of these cases, the problem was mapped so that only nearest neighbor communications were needed. The timings we obtained with data *fetches* carried out using the general router can be interpreted as best case estimates of what one could expect from a CM-2 executor that did not have access to a-priori information on dependency patterns. Table 4 depicts the average time per iteration required to solve a problem over a 64 by 64 by 64 grid, the time was averaged over 100 iterations. Note that in this case there is a roughly eight fold performance difference between the optimized NEWS network program and the version employing the general router fetch instruction. As we mentioned in Section 3, on the 4K processor machine, the program employing the NEWS network achieves a speed of 27.14 MFlops on the matrix vector multiply and a speed of 6.97 MFlops on the triangular solve. The fetch version of the general router program achieves a speed of 3.58 MFlops on the matrix vector multiply and 1.14 MFlops on the triangular solve.

In Table 5 we give comparative timings obtained through the use of the NEWS network and the fetch instruction from the general router for three dimensional meshes with varying edge size. We also present the ratio of the fetch timings to the NEWS net timings. For both the matrix vector multiply and the triangular solve, the ratio between the NEWS net and router timings remain roughly constant for all meshes with edge size up to 64. The router timings become relatively less efficient for the mesh with edge size 128. When we solve this problem with a 128 edge sized mesh, 4 virtual processors are assigned to each actual processor. As described above, the assignment of multiple virtual processors to each actual processor can have the effect of reducing communications overhead; less information needs to be exchanged between processors.

Table 5: Three Dimensional Mesh Sweep and Solve : News Net v.s. General Router Fetch - Varying Mesh Size: 4K processors

Edge Size	MVM NEWS (ms)	MVM Router (ms)	MVM Ratio	Solve NEWS (ms)	Solve Router (ms)	Solve Ratio
16	29	208	7.2	52	328	6.3
32	57	441	7.7	107	664	6.2
64	116	876	7.6	226	1376	6.1
128	478	4581	9.6	961	7098	7.3

5 Conclusion

In Section 3, we presented what we might regard as best case timings for the sparse matrix vector multiplies, sparse triangular solves, and inner products that constitute the iterative portion of Krylov space linear solvers when the solvers use incompletely factored matrices for preconditioning. We performed timings on a large three dimensional model problem over a cube shaped domain discretized with a seven point template. The highest computational rate we were able to achieve for the sparse triangular solve was 13.1 MFlops on 4K processors; this would correspond to a timing of 210 MFlops on an appropriately scaled problem on a 64K processor machine. The highest computational speed we achieved for a matrix vector multiply was 64.2 MFlops; this would correspond to a speed of 1027.0 MFlops in a 64K processor machine. Thus for appropriately structured problems, the CM-2 achieves impressive computational speeds. For both the matrix vector multiply and triangular solve computational kernels, the projected speeds on a full 64K processor machine exceed by a substantial factor the speeds that can be achieved by carefully vectorized code on a single head of a Cray X/MP. These timings appear to leave open the possibility that in large three dimensional problems, the use of incompletely factored matrices as preconditioners in Krylov space methods might speed the solution of some linear systems.

The timings on the CM-2 were highly dependent on the use of very specialized programs. These programs mapped the problem domain onto the processor topology very carefully and used the optimized local NEWS communications network. We explored the consequences of relaxing these restrictions in a variety of ways. Performance degraded very significantly as one abandoned the optimized NEWS communications network and abandoned the careful mapping of a problem domain to the CM-2 processor topology. In some cases we noted 20 to 75 fold degradations in performance as these constraints were relaxed.

The results of our benchmarks clearly demonstrate that one can obtain extremely high performance on Krylov space linear solvers on the CM-2. The benchmarks also

reveal that this performance is unlikely to be achieved in a program capable of handling *general* sparse matrices. It appears that it would be quite feasible to construct a preconditioned Krylov solver capable of solving systems arising from partial differential equations discretized using one of a fixed set of finite difference templates.

References

[1] E. Anderson. *Solving Sparse Triangular Linear Systems on Parallel Computers.* Report 794, UIUC, June 1988.

[2] C. Ashcraft and Roger G. Grimes. *On Vectorizing Incomplete Factorization and SSOR Preconditioners.* Technical Report ETA-TR-41, Boeing Computer Services, December 1986.

[3] D. Baxter, J. Saltz, M. Schultz, and S. Eisenstat. Preconditioned krylov solvers and methods for runtime loop parallelization. In *Proceedings of the 4th International Supercomputing Conf., St. Clara, CA*, April 1989.

[4] D. Baxter, J. Saltz, M. Schultz, S. Eisentstat, and K. Crowley. An experimental study of methods for parallel preconditioned krylov methods. In *Proceedings of the 1988 Hypercube Multiprocessor Conference, Pasadena CA*, January 1988.

[5] H. Berryman, W. Gropp, and J. Saltz. Krylov methods and the cm-2. In *Proceedings of the 4th International Supercomputing Conf., St. Clara, CA*, April 1989.

[6] T. F. Chan, C. C. Kuo, and C. Tong. *Parallel Elliptic Preconditioners: Fourier Analysis and Performance on the Connection Machine.* Report CAM 88-22, UCLA, August 1988.

[7] H. Elman. *Personal Communication.* Report.

[8] J. A. Meijerink and H. A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as occur in practical problems. *Journal of Computational Physics*, 44:134–155, 1981.

[9] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of Computation*, 31:148–162, 1977.

[10] J. Saltz. Aggregation methods for solving sparse triangular systems on multiprocessors. *SIAM J. Sci. and Stat. Computation.*, to appear, 1989.

[11] Herbert L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equatins. *SIAM Journal on Numerical Analysis*, 5:530–558, 1968.

[12] Richard P. Kendall Todd Dupont and H. H. Rachford Jr. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM Journal on Numerical Analysis*, 5:559–573, 1968.

[13] M. Schultz Y. Saad. *Parallel Implementations of Preconditioned Conjugate Gradient Methods*. Department of Computer Science YALEU/DCS/TR-425, Yale University, October 1985.