To appear in the *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, CA, February 1995.

Chapter 1
Preconditioning with a Decoupled Rowwise Ordering on the CM-5*

Sivan Toledo†

Abstract

Decoupled rowwise ordering is an ordering scheme for 2-dimensional grids, which is tailored for preconditioning 5-point difference equations arising from discretizations of partial differential equations. This paper describes the ordering scheme and implementations of a conjugate gradient solver and SSOR preconditioners which use the decoupled rowwise and the red black ordering schemes on the CM-5 parallel supercomputer. The rowwise decoupled preconditioner leads to faster convergence than the red black preconditioner, and it reduces the solution time by a factor of 1.5 to 2.5 over a nonpreconditioned solver on a variety of test problems.

1 Introduction

This paper describes the implementation and performance of a preconditioned conjugate gradient solver for discretized Poisson problems in 2 dimensions, which we have implemented on the Connection Machine CM-5 parallel supercomputer [5]. The preconditioner is an SSOR preconditioner with a novel ordering scheme for the variables, which we call *decoupled rowwise ordering*. Decoupled rowwise ordering, illustrated in Figure 1 and described in detail in Section 2, is tailored for block layouts of rectangular grids, in which each processor in a parallel computer is assigned one rectangular subgrid. The grid points on the boundary of every block are ordered using a red black coloring to decouple the blocks, and interior grid points in every block are ordered row by row. This ordering scheme admits an efficient parallel implementation of the preconditioner on the CM-5 and other distributed memory parallel computers, and it accelerates the solver's convergence more than a red black ordering. The software, which is used in an ocean model being developed by John Marshall and his colleagues at MIT, is described in Section 3, and its performance is discussed in Section 4.

2 Ordering Schemes for Parallel Preconditioners

Preconditioning by Symmetric Successive Over Relaxation (SSOR) amounts to solving two sparse triangular systems, usually by substitution (see for example [1]). The ordering of the variables in the domain has a dramatic effect on both the spectrum of the preconditioned matrix and hence on the convergence rate of iterative solvers, as shown by Duff and

*This research was supported in part by ARPA under Grant N00014–91–J–1698, and by Project SCOUT under ARPA Grant MDA972–92–J–1032. Experiments were performed on CM-5 computers at MIT, Boston University, and NCSA.

†Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, Massachusetts 02139

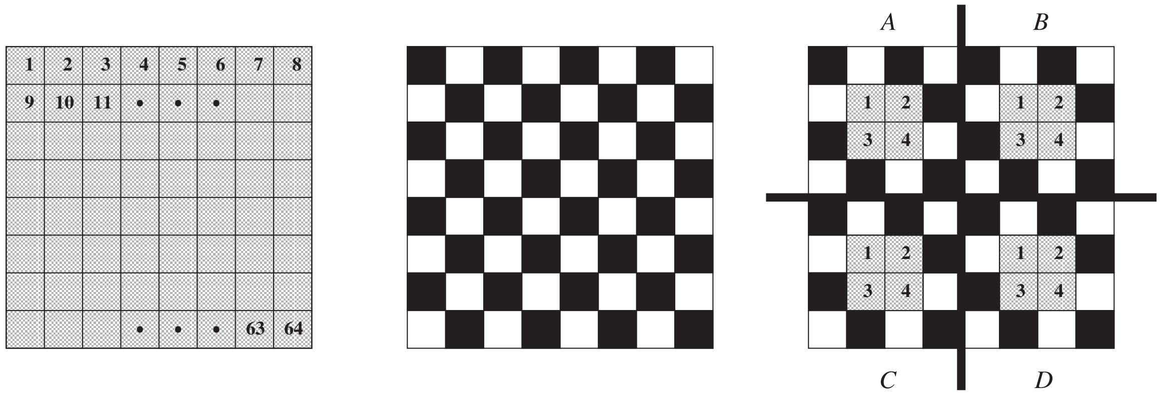

FIG. 1. *Orderings for an 8-by-8 grid. The rowwise ordering is depicted on the left, a red black ordering in the center, and a decoupled rowwise with four blocks marked A, B, C, and D, on the right. The white points are ordered before the black points which are ordered before all interior points. The interior points in each block, which are lightly shaded in the figure, are ordered in rowwise order.*

Meurant [2], and on the implementation of the sparse triangular solvers. This section introduces the decoupled rowwise ordering and compares it with traditional ordering schemes. We focus on preconditioning linear systems of equations $Ax = b$ where A is derived from a 5-point stencil in two dimensions, using a preconditioner of the form $M = LL^{\mathrm{T}}$, where L is a sparse lower triangular matrix, whose sparsity pattern is identical to the sparsity pattern in the lower triangular part of A. In the substitution process for L, the variable that represents point i in the domain depends on the solution of the triangular system at neighboring points that come before i in the ordering, or on neighbors that come after i in the substitution for L^{T}.

The most popular ordering scheme on sequential computers is the rowwise, or natural, ordering. In this ordering scheme the variables are ordered row-by-row, from left to right. The triangular solver for the rowwise ordering contains a significant amount of parallelism — one can solve a system on an n-by-n grid using n processors in $2n$ parallel steps, by proceeding along so-called "wavefronts" [10]. On the CM-5 and similar machines, however, it is hard to exploit this parallelism. In order to balance the computational load, a triangular solver for the rowwise ordering requires a data layout with a long and thin block on every processor, with the Fortran array rotated 45 degrees with respect to the physical domain. Long and thin blocks result in inefficient communication in the rest of the finite-differences computation.

Red black ordering has been proposed as a way of increasing the parallelism in the triangular solver. The variables in the domain are colored red and black in a checkerboard pattern, so that neighboring points always have different colors. All the red points come before all the black points. The resulting triangular solver has a tremendous amount of parallelism — the system can be solved in 2 parallel steps using $n^2/2$ processors. The implementation in data parallel Fortran is simple. Unfortunately, the convergence rate obtained with this ordering is often worse than the convergence rate obtained with the natural ordering.

We propose a new ordering scheme, called decoupled rowwise ordering. Figure 1 depicts the decoupled rowwise ordering along with the rowwise and red black orderings for an 8-by-8 grid. The ordering scheme is based on a decomposition of the grid into rectangular

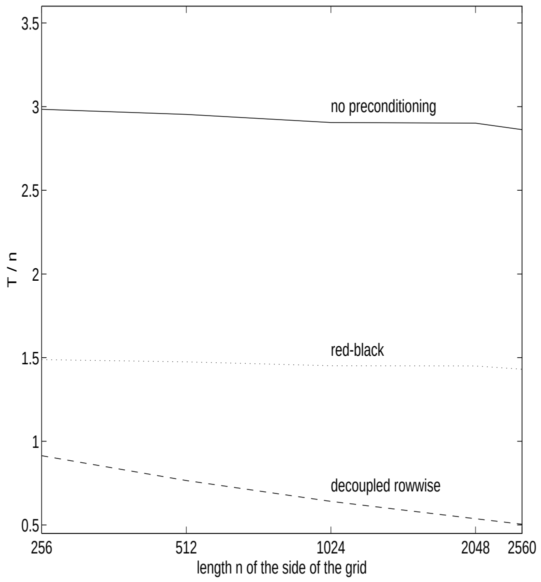

FIG. 2. *Convergence rates of conjugate gradient without preconditioning, with red black preconditioning, and with decoupled rowwise preconditioning. The number of iterations T it took to reduce the 2-norm of the residual by a factor of 10^{-5} divided by the grid length n is plotted as a function of the grid length. The problem is a Poisson equation with Neumann boundary conditions in a square, with a random right-hand side. The decoupled rowwise preconditioner used 128 rectangular blocks.*

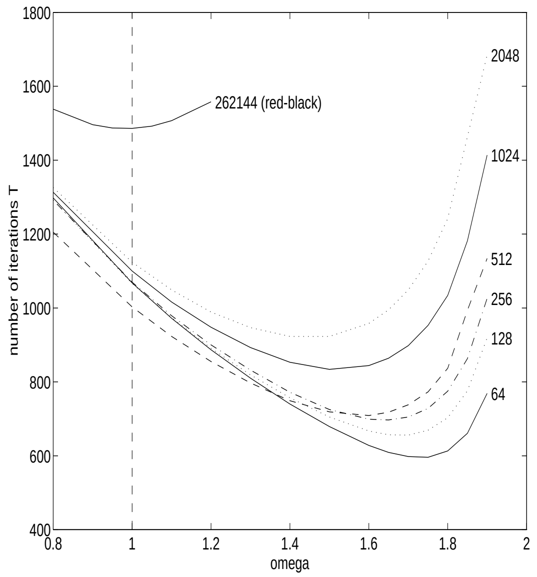

FIG. 3. *Convergence rates of conjugate gradient with decoupled rowwise preconditioning. The number of iterations T it took to reduce the 2-norm of the residual by a factor of 10^{-5} is plotted as a function of the over relaxation parameter ω for a number of block sizes. The number next to each graph is the number of blocks to which the square domain is decomposed. The problem is a Poisson equation with Neumann boundary conditions in a square discretized on a* 1024*-by-*1024 *grid, with a random right-hand side.*

subgrids. The boundary points of every block are colored red black and placed first in the ordering, with the red boundary points before the black boundary points before all interior points. The interior points in every block are ordered in a rowwise order. The triangular solver solves first for the red points which depend on no other points, then for the black points which depend on their red neighbors, and then for the interior points. Solving for the boundary points, which can be done in 2 parallel steps, completely decouples the blocks, and each processor can proceed to solve for all the interior points in its block. Decoupled rowwise ordering scheme is somewhat similar to the ordering scheme called `vdv1` in [2], but `vdv1` does not decouple the blocks in the same way, and is therefore less suitable for parallel implementations. The decomposition of the grid used by our software is simply the block layout of the array representing the grid in the parallel computer.

On our test problems, the decoupled rowwise preconditioner accelerates the convergence more than the red black preconditioner. Figure 2 shows the convergence rate of conjugate gradient without preconditioning, with red black preconditioning, and with decoupled rowwise preconditioning, for a Poisson equation with Neumann boundary conditions and a random right-hand side. The preconditioners use an optimal over relaxation parameter ω. The optimal value of ω was determined experimentally for each problem and preconditioner.

The number of iterations required to reduce the norm of the residual by a given factor is proportional to the length of the the grid side for the nonpreconditioned solver and for the red black preconditioning. For the decoupled rowwise preconditioner, however, the number of iterations depends sublinearly on the grid-side length, and it is always smaller than the number of iterations required by the red black preconditioner. The decoupled rowwise preconditioner reduces the number of iterations by a factor of 3.4–5.4 over the number required by the nonpreconditioned solver, whereas the red black preconditioner reduces the number of iteration by a factor of roughly 2. The preconditioners behave in a similar manner on other test problems, such as problems in nonsquare domains with holes and problems with Dirichlet boundary conditions.

Figure 3 shows the convergence rate of the solver with the decoupled rowwise preconditioner as a function of the over relaxation parameter ω for a range of block sizes. For a given problem, orderings with few large blocks generally lead to faster convergence than orderings with many small blocks. Although this rule does not always hold, in our experiments it always holds for the optimal value of ω. Figure 3 also demonstrate that the optimal value of ω decreases as the number of blocks increases. In the limit, when the length of the side of each block is 1 or 2, decoupled rowwise degenerates into a red black ordering, in which the optimal value of ω is very close to 1 in our experiments.

3 Implementation

This section describes our implementation of the conjugate gradient solver and the preconditioners on the CM-5. The software includes a conjugate gradient solver, with a matrix-vector multiply and preconditioners for 5-point stencils in 2 dimensions. The stencils use one variable coefficient and four constant coefficients.

We have implemented the solver on the CM-5 using a combination of CM Fortran [9] and CDPEAC [7, 8], the assembly language of the vector units with which the CM-5's nodes are equipped. The overall solver is written in Fortran, the nodal part of the sparse matrix-vector multiply is implemented in CDPEAC, and so is the nodal part of the two preconditioners. The communication in both the matrix-vector multiply and the preconditioners is done in Fortran, using shift operations on small "ghost" arrays into which the boundary grid points on every block are copied. The copying from the grid to the ghost arrays is done in CDPEAC, however, since the CM Fortran compiler does not vectorize these operations well. The CDPEAC subroutine that computes the nodal part of the matrix-vector multiply also computes the inner product of the vector before and after the multiplication, which is required in the conjugate gradient method. The integration of the inner product into the matrix-vector multiply eliminates the need to load the two vectors into registers again. The red black preconditioner uses strided load and store instructions to select red or black points. The CDPEAC subroutines consist of more than 1500 lines of code. The software is available by anonymous FTP from `theory.lcs.mit.edu` in the directory `pub/sivan`.

We wrote the solver partially in assembly language for three reasons. First, we wanted to obtain the highest quality solver we could, and we could not achieve this goal with either Fortran or CMSSL [6], the CM-5 Scientific Subroutine Library, as explained below. Second, since the ordering scheme requires using DO loops within each block, or on each node, there is no way to efficiently implement the preconditioner in Fortran. The only possible way to express such a computation in CM Fortran or High Performance Fortran is using a DO loop on serial axes, but this does not compile on the CM-5 into a tight loop on the nodes, but rather into a loop containing a broadcast in every iteration, whose performance is

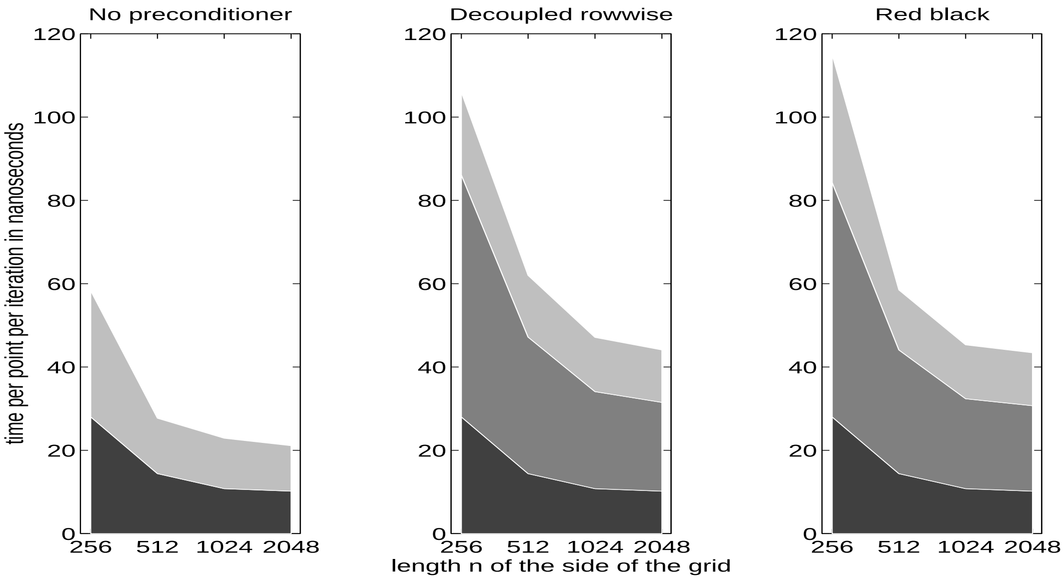

FIG. 4. *The running time of the solver per point per iteration as a function of the problem size, on a 32-node CM-5. The bottom, darkest section in each graph represents the matrix-vector multiplication time, the middle section represents the preconditioning time (if a preconditioner is used), and the top and lightest section represents the rest of the vector operations.*

about two orders of magnitude worse than our CDPEAC implementation. Third, since the usefulness of a preconditioner depends on the ratio between the preconditioning time and the conjugate gradient iteration time, the only way to evaluate the preconditioner is using the best possible implementation for both the preconditioner and the conjugate gradient solver.

4 Performance

To put the performance of our solver in perspective, we compare the performance of our sparse matrix-vector multiply subroutine with the performance of a subroutine written in CM Fortran and a subroutine from the Connection Machine Scientific Subroutine Library (CMSSL)[1] with the same functionality (CMSSL only contains variable-coefficients stencil matrix-vector multiply, which is the subroutine we use in the comparison). On grids with 2K points per node, the CM Fortran version is 71% slower than our version, whereas the CMSSL version was 45% slower than our version. On larger grids with 128K points per node, the CM Fortran version is 96% slower and the CMSSL version is 55% slower than our version.

Figure 4 shows the running time of the solver per point per iteration, measured on a 32-node CM-5. The graphs show the matrix-vector multiplication time, the preconditioning time (for both a red black and a decoupled rowwise preconditioners), and the overall time, for problems sizes ranging from 256-by-256 up to 2048-by-2048. Without preconditioning the iteration time is divided roughly evenly between the matrix-vector multiplication and the rest of the vector operations in the solver, and both preconditioners roughly double the running time of each iteration.

[1] We have used CM Fortran version 2.1.1-2 Final and CMSSL version 3.1 Final.

5 Conclusions

We have presented an efficient conjugate gradient solver for 5-point difference equations in 2 dimensions, utilizing an SSOR preconditioner with a novel ordering scheme for the variables. Thanks to careful coding and the use of assembly language, the implementation is more efficient than CM Fortran and CMSSL implementations, and the preconditioner is about as efficient as a high-performance red black preconditioner. The preconditioner accelerates the solver's convergence more than a red black preconditioner, leading to speedups by a factor of 1.5 to 2.5 over the nonpreconditioned solver.

We believe that the form and the amount of parallelism in an algorithm must match the architecture of the machine which executes it. In our case, the parallelism in the rowwise decoupled ordering is easier to exploit on the CM-5 than the wavefront parallelism in the rowwise ordering, and the amount of parallelism is smaller than the parallelism in a red black ordering. The smaller amount of parallelism is probably an advantage, since it seems to improve the convergence rate while still admitting an efficient parallel implementation on hundreds of processors.

It is important that programming languages, especially High Performance Fortran [3, 4], allow programmers to express algorithms in which serial execution on each node is used. This could be accomplished either via compiler optimizations of expressions involving arrays with serial axes, or by allowing the programmer to write nodal code, which is not currently possible in CM Fortran[2]. We prefer the second option since it is simpler for both the programmer and the compiler writer. Writing sequential nodal code does not necessitate the use of message passing libraries, since the communication can be done in data-parallel Fortran using ghost arrays.

Acknowledgments

I would like to thank Chris N. Hill for his efforts to incorporate the preconditioner into the MIT Global Circulation Model, and to Robert Putnam for executing a set of experiments on Boston University's CM-5.

References

[1] R. Barrentt, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1993.

[2] I. S. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradient*, BIT, 29 (1989), pp. 635–657.

[3] High Performance Fortran Forum, *High Performance Fortran language specification*, Scientific Programming, 2 (1994).

[4] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr., and M. E. Zosel, *The High Performance Fortran Handbook*, The MIT Press, 1994.

[5] Thinking Machines Corporation, *The Connection Machine CM-5 Technical Summary*, Cambridge, MA, Jan. 1992.

[6] ——, *CMSSL for CM-Fortran*, Cambridge, MA, June 1993.

[7] ——, *DPEAC Reference Manual*, Cambridge, MA, Mar. 1993.

[8] ——, *VU Programmer's Handbook*, Cambridge, MA, Aug. 1993.

[9] ——, *CM-Fortran Language Reference Manual*, Cambridge, MA, Jan. 1994.

[10] H. van der Vorst, *(M)ICCG for 2D problems on vector computers*, in Supercomputing, A. Linchnewsky and C. Saguez, eds., North Holland, 1988.

[2] It is possible to write nodal subroutines in Fortran 77 on the CM-5, but they do not utilize its vector units.