Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

THE PRECONDITIONED CONJUGATE GRADIENT METHOD ON THE CONNECTION MACHINE

Charles Tong

October 1988
CSD-880077

The Preconditioned Conjugate Gradient Method on the Connection Machine[*]

Charles Tong †

September, 1988

Abstract

This paper presents the parallel implementations of the preconditioned conjugate gradient method on the Connection Machine. A number of preconditioners, including the incomplete LU (ILU) factorization with natural ordering, modified ILU (MILU) factorization with natural ordering, symmetric successive overrelaxation (SSOR) with natural ordering, ILU with red/black ordering, MILU with red/black ordering, SSOR with red/black ordering, and polynomial preconditioners, were evaluated based upon the convergence rates and execution times on the Connection Machine.

Key Words : Connection Machine (CM), preconditioned conjugate gradient (PCG) method, incomplete LU (ILU) factorization, Modified incomplete LU (MILU) factorization, Symmetric successive overrelaxation (SSOR), polynomial preconditioner, diagonal natural ordering, red/black ordering.

---

* This work was supported in part by the Department of Energy under contract DE-FG03-87ER25037, the National Science Foundation under contracts NSF-DMS87-14612 and BBS 87 14206, and the Army Research Office under contract DAAL03-88-K-0085.

† Department of Computer Science, University of California, Los Angeles, CA 90024.

1. Introduction

The conjugate gradient method, coupled with "good" preconditioning, has been known as an efficient technique for solving large sparse symmetric positive definite linear systems of equations such as those generated by the discretization of elliptic partial differential equations in two or three dimensions. In the past, many preconditioners have been proposed which helped to make the PCG method very competitive for computer implementation. Many of such preconditioners, however, are sequential in nature; and thus are unable to exploit efficiently the computational resources offered by massively parallel computers. In search for more parallel preconditioners, one method is to re-order the unknowns in the system of equations so that the data dependency can be reduced, and higher degree of parallelism can be obtained. Nevertheless, experiments and analyses have shown that more parallel preconditioners often give slower convergence rates than their sequential counterparts, The conflict between convergence rates and degree of parallelism for a preconditoner should prompt researchers to look for preconditoners that offer high degree of parallelism while preserving fast convergence rates to achieve the best overall computation time performance.

This paper presents the results of implementation of a number of preconditioners on the Connection machine. Among these preconditioners are : incomplete LU (ILU) factorization with natural ordering, Modified incomplete LU (MILU) factorization with natural ordering, symmetric successive overrelaxation (SSOR) with natural ordering, ILU factorization with red/black ordering, MILU with red/black ordering, SSOR with red/black ordering, and a few polynomial preconditioners. Section 2 will cover the basic conjugate gradient method, different types of orderings, as well as a number of preconditioners. Section 3 will present a brief description of the Connection Machine used in this experiment. Section 4 will discuss the detail of implementation. Finally, in section 5, both the convergence rates data as well as execution times on the Connection Machine for different preconditioners will be presented.

2. The Preconditioned Conjugate Gradient Methods

2.1 The Preconditioned Conjugate Gradient (PCG) Algorithm

The PCG algorithm for the solution of a large sparse symmetric positive definite linear system of equations

$$A\,u = f$$

where A is an $N \times N$ symmetric positive definite matrix and u and f are $N \times 1$ vectors, is given as follow [11] :

```
$r = f - A\,u$                              ; initial residual
$p = 0$
Repeat
    $z = M^{-1}\,r$                         ; preconditioning
    $\beta = new <r, z> / old <r, z>$
    $p = z + \beta\,p$                      ; updating direction
    $\alpha = new <r, z> / <p, Ap>$
    $u = u + \alpha\,p$                     ; updating solution
    $r = r - \alpha\,Ap$                    ; updating the residual
until $<r, r> \le tolerance$
```

where $<\cdot, \cdot>$ denotes the usual Euclidean inner product, and r and p are $N \times 1$ residual and search direction vectors respectively.

The matrix M is called the preconditioning matrix and the speed with which the algorithm converges depends strongly on the choice of M. It is desirable to have M approximating A so that the condition number $\kappa\,(M^{-1}\,A)$ is smaller than that of A alone, that M retains the sparsity feature, and that the computational overhead to solve the system of equations

$$M\,z = r$$

is relatively small.

2.2 Model problems and orderings

The model problem used in this experiment is the discrete Poisson equation on the unit square $\Omega = [0,1]^2$ with Dirichlet boundary condition. The 5-point finite difference approximation of the problem is :

$$u_{j+1,k} + u_{j-1,k} + u_{j,k+1} + u_{j,k-1} - 4u_{j,k} = f_{j,k}\,h^2 \qquad j,k = 1, \cdots, n-1 \quad where\ n = \frac{1}{h},$$

where h is the grid spacing in both the x and y directions ,and $u_{j,k}$ is used to approximate the value of $u\,(jh,kh)$. In terms of the shift operators, the problem can be rewritten as :

$$A_{j,k}\,u_{j,k} = -\,\frac{h^2\,f_{j,k}}{4}\,, \quad A_{j,k} = 1 - \frac{1}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,)\,,$$

where E_x and E_y are shift operators along the x and y directions, such that

$$E_x\, u_{j,k} = u_{j+1,k} \;,\; E_x^{-1}\, u_{j,k} = u_{j-1,k} \;,\; E_y\, u_{j,k} = u_{j,k+1} \;,\; E_y^{-1}\, u_{j,k} = u_{j,k-1} \,.$$

Therefore, $A_{j,k}$ is a local operator at the grid point (jh,kh) [1]. A stencil representation of operator $A_{j,k}$ is described in Figure 1. A collection of local operators $A_{j,k}$ at all grid points, together with the way that the grid points are ordered, form the coefficient matrix A.

The ordering of grid points on a 2-dimensional grid determines the form of the coefficient matrix A and also that of the preconditioners. Using the natural ordering, grid points are ordered in row-wise (or column-wise) manner. And using the red/black ordering, grid points are first partitioned into red and black groups such that a grid point (j,k) is red if $j+k$ is even and black if it is odd. Then the grid points within red group are ordered, followed by the ordering of the grid points within the black group. In the context of parallel computation, the natual and red/black orderings evolve into their parallel versions, which are called diagonal and parallel red/black orderings respectively. These two (partial) orderings are defined as follow :

Diagonal ordering :

$$(j,k) < (m,n) \qquad \text{if } j+k < m+n,$$

Parallel Red/black ordering :

$$(j,k) < (m,n) \qquad \text{if } (j,k) \textit{ is red} \text{ and } (m,n) \textit{ is black},$$

where the order of updates during preconditioning is determined by the inequality condition (e.g. in ascending or descending order). These two orderings for the grid points on a uniform 6×6 square grid are illustrated in Figure 2. For the diagonal ordering, the grid points that have the same sum $j+k$ can be performed in parallel, and the same is true for the red/black ordering where the grid points are of the same color. This means that if this problem is solved on a parallel computer, then a sweep (or one iteration) using red/black ordering can at best be computed in constant time (independent of the number of grid points) while a sweep using diagonal ordering can at best be computed in $O(N^{1/2})$ time, where N is the number of grid points. Here we can see that the parallel red/black ordering offers higher degree of parallelism ($O(N)$ operations can be performed in parallel) than the diagonal ordering (which has degree of parallelism $O(N^{1/2})$). Nevertheless, the convergence rates improvement of the preconditioners using diagonal ordering are usually better than those using parallel red/black ordering.

2.3 Preconditioners

The preconditioners under investigation in this experiment are described in the following sub-sections.

2.3.1 Incomplete LU (ILU) preconditioners [1,3,12]

2.3.1.1 ILU with diagonal ordering

The ILU preconditioner is defined to be the product $M_I = LU$, where L and U are lower and upper triangular matrices respectively, and U has unit diagonal. The L and U are constructed such that the entries of M_I have the same values as those of A wherever the corresponding entry in A is non-zero. For model problems such as 2-D Poisson equation with Dirichlet periodic boundary conditions, one way is to perform LU factorization on the M matrix to obtain the exact local operators. An easier way is to get an matrix which is an approximate of M so that following local operators $L_{j,k}$ and $U_{j,k}$ can be used on all grid points (this easier method is implemented since it was found that there is no convergence rate loss by using this method) :

$$L_{j,k} = \frac{1}{4} (a - E_x^{-1} - E_y^{-1}) , \quad U_{j,k} = 1 - \frac{1}{a} E_x - \frac{1}{a} E_{y,}$$

where a is a constant to be determined. The product of $L_{j,k}$ and $U_{j,k}$ is thus :

$$M_{I\,(j,k)} = \frac{1}{4} [a + \frac{2}{a} - (E_x + E_y + E_x^{-1} + E_y^{-1}) + \frac{1}{a} (E_x E_y^{-1} + E_x^{-1} E_y)]$$

By matching the entries of M and A according to the requirement formulated above (again, since an approximate M is used, the requirements are not obeyed for a few rows), the following condition is obtained :

$$a + \frac{2}{a} = 4,$$

The solutions of this equation are $a = 2 \pm \sqrt{2}$. We choose $a = 2 + \sqrt{2}$, since the coefficients of the corresponding difference operator $R_I = M_I - A$ have smaller absolute values. Thus, the diagonal-ordered ILU preconditioning consists of a forward solve (L^{-1}) followed by a backward solve (U^{-1}). Due to the sequential nature of the diagonal ordered ILU preconditioning, these forward and backward solves can at best be performed in $O(\sqrt{N})$ time on parallel computers.

2.3.1.2 ILU with parallel red/black ordering

The local operators $L_{j,k}$ and $U_{j,k}$ for the parallel red/black ordering are :

$$L_{j,k} = \begin{cases} 1 , & (j,k)\ red \\ 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,) , & (j,k)\ black \end{cases}$$

$$U_{j,k} = \begin{cases} 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,) , & (j,k)\ red \\ \frac{3}{4} , & (j,k)\ black \end{cases}$$

Therefore, we obtain the parallel red/black ordered ILU local operator as :

$$M_{Irb\,(j,k)} =$$

$$\begin{cases} 1 - \frac{1}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,) , & (j,k)\ \ red \\ \frac{3}{4} - \frac{1}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,) + \frac{1}{16}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)^2 , & (j,k)\ \ black \end{cases}$$

Again, this preconditioning consists of a forward solve (L^{-1}) followed by a backward solve (U^{-1}). In this case, however, since all the red points can be updated in parallel and so can the all the black points, the solves can at best be done in constant time. The stencil representations of the local operators for the diagonal-ordered and parallel red/black ordered ILU preconditioners are shown in Figure 3.

2.3.2 MILU preconditioners [1,3,13]

2.3.2.1 MILU with Diagonal Ordering

The MILU preconditioner with diagonal ordering is defined so that the entries of M_M have the same values as the corresponding off-diagonal entries of A which are nonzero, and the sum of the entries of each row of the error matrix $R_M = M_M - A$ equals $\delta = ch^2$, where h is the grid spacing and c is a nonnegative constant that is independent of h (as in the ILU case, only an approximate M is used in this experiment, which means that the row sum criterion is violated for a few rows). Thus, here again :

$$M_{M\,(j,k)} = \frac{1}{4}\,[\,a + \frac{2}{a} - (\,E_x + E_y + E_x^{-1} + E_y^{-1}\,) + \frac{1}{a}\,(\,E_x\,E_y^{-1} + E_x^{-1}\,E_y\,)\,]$$

Again, equating the entries of M_M and A according to the requirement, we have :

$$a + \frac{4}{a} - 4 = \delta,$$

By solving the above equation, we obtain :

$$a = 2 + \frac{\delta}{2} + \frac{1}{2}\sqrt{8\,\delta + \delta^2}$$

The steps involved in this preconditioning are the same as those of the ILU with diagonal ordering. Hence, this preconditioning can at best be completed in $O(\sqrt{N})$ time on parallel computers.

2.3.2.2 MILU With parallel red/black ordering

The local operators $L_{j,k}$ and $U_{j,k}$ for the parallel red/black ordering are :

$$L_{j,k} = \begin{cases} 1 + \delta, & (j,k)\ red \\ 1 + \delta - \dfrac{1}{1+\delta} - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ black \end{cases}$$

$$U_{j,k} = \begin{cases} 1 - \dfrac{1}{4\,(1+\delta)}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ red \\ 1, & (j,k)\ black \end{cases}$$

where δ was defined in the previous sub-section.

Again, we obtain the parallel red/black ordered MILU preconditioner as :

$M_{Mrb\,(j,k)} =$

$$\begin{cases} 1 + \delta - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ red \\ 1 + \delta - \dfrac{1}{1+\delta} - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}) + \dfrac{1}{16\,(1+\delta)}\,(E_x + E_x^{-1} + E_y + E_y^{-1})^2, & (j,k)\ black \end{cases}$$

This preconditioner, as in ILU preconditioner with parallel red/black ordering, takes constant time independent of the number of grid points, N, on parallel computers. The stencil representations of the local operators for the diagonal-ordered MILU is the same as that for the ILU case, and those for the parallel red/black ordered MILU preconditioner are shown in Figure 4.

2.3.3 Symmetric Successive Overrelaxation (SSOR) precondtioners [1,3,14]

2.3.3.1 SSOR with diagonal ordering

The SSOR preconditioner is defined to be :

$$M_S = (D - \omega L) D^{-1} (D - \omega L^T),$$

where D and L are diagonal and strictly lower triangular matrices respectively such that :

$$A = D - L - L^T ,$$

and ω is the relaxation parameter.

For the model Poisson problem with diagonal ordering, the partitioning leads to the following local operators :

$$D_{j,k} = 1 , \quad L_{j,k} = \frac{1}{4} (E_x^{-1} + E_y^{-1}), \quad L_{j,k}^{-T} = \frac{1}{4} (E_x + E_y) .$$

The product $M_{S\,(j,k)}$ is thus :

$$M_{S\,(j,k)} = 1 - \frac{\omega}{4} (E_x + E_y + E_x^{-1} + E_y^{-1}) + \frac{\omega^2}{16} (2 + E_x^{-1} E_y + E_x E_y^{-1}) .$$

Again, as in ILU and MILU preconditioners with diagonal ordering, this preconditioning also takes $O(\sqrt{N})$ time on parallel computers.

2.3.3.2 SSOR With parallel red/black ordering

For the model Poisson problem with parallel red/black ordering, the partitioning leads to the following local operators :

$$D_{j,k} = 1 ,$$

$$L_{j,k} = \begin{cases} 0 , & (j,k)\ red \\ 1 - \frac{1}{4} (E_x + E_x^{-1} + E_y + E_y^{-1}) , & (j,k)\ black \end{cases}$$

$$U_{j,k} = \begin{cases} 1 - \frac{1}{4} (E_x + E_x^{-1} + E_y + E_y^{-1}) , & (j,k)\ red \\ 0 , & (j,k)\ black \end{cases}$$

Therefore, we obtain the parallel red/black ordered SSOR preconditioner as :

$M_{Srb\,(j,k)} =$

$$\begin{cases} 1 - \frac{\omega}{4} (E_x + E_x^{-1} + E_y + E_y^{-1}) , & (j,k) \ red \\ 1 - \frac{\omega}{4} (E_x + E_x^{-1} + E_y + E_y^{-1}) + \frac{\omega^2}{16} (E_x + E_x^{-1} + E_y + E_y^{-1})^2 , & (j,k) \ black \end{cases}$$

The stencil representations of the local operators for the diagonal-ordered and parallel red/black ordered SSOR preconditioners are shown in Figure 4.

2.3.4 Polynomial preconditioners [1,4,6]

The polynomial preconditioner attempts to approximate the inverse of $A = (I - B) P^{-1}$ matrix by truncating its Neumann series expansion,

$$A^{-1} = P (I - B)^{-1} \approx P (I + B + B^2 + \cdots + B^{m-1}) = M_{P\,(m)}^{-1} ,$$

where, in operator form, $B = E_x + E_x^{-1} + E_y + E_y^{-1}$ and we can use $M_{P\,(m)}$ as the polynomial preconditioner. This preconditioner will be called m-step Jacobi preconditioner in the subsequent sections.

In general, we can consider the polynomial preconditioner as :

$$M_{GP\,(m)}^{-1} = \sum_{l=0}^{m} \gamma_l B^l ,$$

so that the coefficients γ_l, $0 \le l \le m$, can be chosen to minimize the condition number of the preconditioned system $M_m^{-1} A$ for fixed m, or minimize the mean square error with respect to some weight functions [4].

The polynomial preconditioners formulated above is a very good candidate for parallel computation. If the P matrix is the identity matrix, then the $m - 1$ steps for the m-step Jacobi preconditioning amount to $m - 1$ iterations of the basic Jacobi method followed by accumulating the results of the iterations. As the basic Jacobi method offers a very high degree of parallelism (all grid points can be updated at the same time), so is this m-step Jacobi preconditioning. Thus, on massively parallel computer systems such as the CM, the m-step Jacobi preconditioning takes $O(m)$ time.

2.4 Convergence Rates

The convergence rates of the conjugate gradient method with different preconditioners for the model Poisson problem, which depend on both the corresponding condition number as well as the distribution of the eigenvalues of the preconditioned system, can be studied either by matrix iterative analysis or by Fourier analysis. The result of Fourier analysis on the preconditioners mentioned above is presented in the following table [1].

preconditioner	Convergence rates
basic CG	$O(N^{0.5})$
ILU (diagonal)	$O(N^{0.5})$
MILU (diagonal)	$O(N^{0.25})$
SSOR (diagonal)	$O(N^{0.25})$
ILU (R/B)	$O(N^{0.5})$
MILU (R/B)	$O(N^{0.5})$
SSOR (R/B)	$O(N^{0.5})$
m-step Jacobi	$O(N^{0.5})$

Table 1 : the theoretical convergence rates for different preconditioners

3. The Connection Machine

The Connection Machine (CM) is a massively parallel computer consisting of 65536 single bit processors. The CM system consists of two parts - a front end machine and a hypercube of 64k processors. The front end computers, currently supported by the Symbolics and the VAX machines, provides instruction sequencing, program development, and low speed I/O. The CM programs contain two types of statements - those operating on single data items and those operating on whole data sets at once. The single-data-item instructions are executed in the front end, whereas the large-data-set instructions are executed in the CM hypercube for parallel processing.

The CM hypercube consists of 4096 (2^{12}) chips, each with 16 processors and a router, to form a 12 dimensional hypecube. The router is a communication processor that allows any on-chip processor to communicate with any other processors in the system. In addition to the router hypercube network there is a separate communication facility called the NEWS grid. That is, each processor is wired to its four nearest neighbors in a two-dimensional regular grid. Machine instructions allow message to be sent, a bit at a time, to the processor to the north, south, east or west on this grid. Communication on this NEWS grid is very fast compared to the router communication and is encouraged for short distance communication between processors. Moreover, on the CM-2, there are 8k bytes of random access memory associated with each physical processor; and also the Weitek floating point processors are incorporated into the system, with one Weitek chip shared between 2 16-processor CM chips.

An important feature of the CM system is its support for virtual processors. A run-time configuration command *cold-boot* may be used to specify how each physical processor is to simulate a small rectangular array of virtual processors - except that such virtual processors appear to be correspondingly slower and have only a fraction of the memory of a physical processor.

The CM supports three software environments : Common LISP, Fortran, and C. In each case, a standard extension to the language is provided that supplies the complete interface to the hypercube. The extensions are *LISP, CM Fortran, and C* respectively. The basic idea in each case is that scalar variables reside on the front end, while vector of parallel variables are distributed over the cube [5].

4. Implementation

4.1 The Connection Machine

The CM used in the present experiment is a 16k-node CM-2 without the Weitek floating point hardware running at a clock frequency of about 6.47 MHz. The front end computer used is the Symbolics machine and the language used for program development is *LISP.

4.2 Processor mapping

The model 2D Poisson problem was mapped onto the 2D NEWS grid of the Connection Machine so that each grid point is mapped to by a different processor. For example, to run a 128 × 128 grid problem, the following configuration command can be used :

(**cold-boot :initial-dimensions* '(128 128))

In this case, since there are altogether $128 \times 128 = 16384$ grid points and we have 16384 (16k) physical processors, each physical processor can perform the computations for a single grid point. Suppose 65336 (256×256) grid points are to be simulated but the same number of physical processors (16k) are available, then each physical processor has to take the computation load of 4 grid points. By using the virtual processing capability of the CM, this mapping (the mapping of 4 grid points to one physical processor) is transparent to the users and is performed automatically when the

(**cold–boot* *:initial–dimensions* '(256 256))

statement is executed.

An advantage of mapping the problem on the 2D NEWS grid is that the neighboring grid points are mapped to neighboring processors; and since the communication overhead between neighboring processors using the NEWS communication is extremely fast, and that most of the computations are between local grid points, good total execution time performance can be expected.

One major concern is that if each processor simulates one grid point and red/black ordering is used, then only half of the processors will be active during updating the black or the red grid points, resulting in low processor utilization. It happens that the virtual processing capability on the CM can handle this problem elegantly : if the number of grid points is less than or equal to the number of physical processors available, then some processors will be idle in any case, and there is no way to improve the utilization of the processors. Suppose the nu;mber of grid points is 4 times as many as the number of physical processors, then by using the **cold–boot* as described above, two black and two red points will be mapped to each physical processor. Here we can see that the physical processors will be kept busy all of the time, computing either black or red points.

Another concern is how the boundary grid points are handled. Since no computation is needed for the boundary grid points other than providing data to their neighbors, one way is not to map them to any processors and the neighbors of these boundary grid points are maped to the boundary processors. During computation, these boundary processors will be performing slightly different task from the other interior processors. An example is the local operator $A_{j,k}$ which requires fetching data from four neighbors (to the north, south, east, and west). Since the boundary processors have one or two neighbors missing (e.g. the processor at the upper right hand corner will not have neighbors to its east and north directions), they have to execute this operator a little differently. Because the CM is a SIMD (single-instruction-multiple-data) machine and cannot execute two different active operations simultaneouly, the updating of interior and boundary processors have to be done in two separate steps, resulting in longer execution time. Another way is to map boundary grid points also to actual processors. The drawback to this scheme is that these boundary processors will be idle most of the time. However, since the operations to be performed on all processors will be the identical, the updating takes only one step, resulting in shorter execution time compared to the first scheme.

This latter scheme is chosen for our implementation for reasons that it is simple to implement and it will probably give better performance.

To distinguish between the boundary processors and the interior ones, a boolean variable, *grid−interior−flag*, is declared which is set to true for the interior processors, and false for the boundary processors. Every time computation is to be performed only on the interior processors (or grid points), the following *LISP statement can be used to achieve the desired results.

```
(*when grid-interior-flag
    (do something to the grid points)
)
```

4.3 The preconditioned conjugate gradient method on the CM

One iteration of PCG method requires 3 inner products (including the residual calculation), 3 multiply-and-add operations, 1 matrix-vector product calculation, 2 scalar divisions, one comparison, plus the calculation required for preconditioning. Let's look at how each of these operations is performed on the CM :

a. The multiply-and-add operation

This operation is in the form of $y=ax+b$ where x and b are vectors and a is a scalar. If each processor takes care of one element in the vector and the scalar a is supplied by the host system, then the tasks performed by each processor are first to receive the a, then multiply a by the x which is stored within the processor, and lastly add to the product the variable b which is also stored within the processor. The *LISP version of this operation is :

```
(*when grid-interior-flag
    (*set pvar-y (+!! (*!! pvar-x (!! a)) pvar-b))
)
```

The time to perform this operation depends only on the virtual processor ratio (number of grid points per physical processor) and this ratio depends on the total number of grid points and the total number of available processors. For a particular virtual processor ratio, this operation takes constant time.

b. The matrix-vector product ($ap = A\ p$)

The matrix in this case is A, which, when operating on a vector, is equivalent to parallel execution of the local operator $A_{j,k}$ (defined previously) on each element of the vector. As described in Figure 1, each processor adds the data fetched from the processors to its north, south, east, and west, divides the sum by 4, and subtracts the quotient from its own data. In *LISP code, it can be represented as :

```
(*when grid-interior-flag
    (*set pvar-ap
        (-!!
            pvar-p
            (/!!
              (+!!
                  (pref-relative-grid!! pvar-p (!! 1) (!! 0))
                  (pref-relative-grid!! pvar-p (!! 0) (!! 1))
                  (pref-relative-grid!! pvar-p (!! -1) (!! 0))
                  (pref-relative-grid!! pvar-p (!! 0) (!! -1))
              )
              (!! 4.0)
            )
        )
    )
)
```

where the 'pref-relative-grid' function has 3 arguments - the variable to be fetched from the destination processor, the relative distance of the destination processor in the x-direction, and the relative distance of the destination processor in the y-direction respectively.

Again, this operation can be done in constant time for a particular virtual processor ratio.

c. Inner Product

The inner product has been known as a bottleneck to the performance of PCG method. It is important that this inner product operation can be done efficiently. It can be observed that the hypercube configuration of the CM helps to speed up the inner product computation. It allows multiplication to be done in parallel in all processors, and the the partial sums are accumulated in the form of a binary tree. The execution time of this inner product is thus $O(\log N)$ where N is the total number of grid points. The *LISP code for inner product calculation is :

```
(*when grid-interior-flag
    (setq inner-product (*sum (*!! pvar-r pvar-r)))
)
```

d. Others

Other computational needs include 2 scalar divides and 1 comparison for convergence. These computations are performed on the host and require constant time.

In summary, without considering the preconditioning, the overall computation time for each iteration is dominated by the inner product operation and thus takes $O(\log N)$.

For preconditioning, depending on whether diagonal ordering or red/black ordering is used, the order of computation times can be quite different. For diagonal ordering, since there are $O(\sqrt{N})$ diagonal in a 2D grid, the corresponding preconditioning takes $O(\sqrt{N})$ time. For red/black ordering, since all the red points can be updated in parallel, and so are the black points, the corresponding preconditioning only takes constant time.

Thus, depending on whether diagonal ordering or red/black ordering is used, the order of execution times per iteration are $O(\sqrt{N})$ and $O(\log N)$ respectively.

4.4 Experiments

In this experiment, the 2-D Poisson equation with Dirichlet boundary condition was solved using the PCG method. Specifically, the Poisson equation,

$$u_{xx} + u_{yy} = 4$$

is used and the boundary conditions are :

$$u(x,y) = x^2 + y^2 \qquad \text{for } \textit{the boundary grid points.}$$

The PCG method was run on grid spacing (h) of $1/7$, $1/15$, $1/31$, $1/63$, and$1/127$ (for some preconditioners, grid spacing of $1/255$ is also experimented). The initial guess was 0.0, and the number of iterations needed to reduce the residual to less or equal to 10^{-4} as well as the corresponding CM execution times were recorded and plotted. The experiments being run are as follow :

1. Conjugate gradient method without preconditioning (CG),

2. PCG - ILU with diagonal ordering (ILU diagonal),
3. PCG - MILU with diagonal ordering (MILU diagonal), also performance as a function of c,
4. PCG - SSOR with diagonal ordering (SSOR diagonal),
5. PCG - ILU with red/black ordering (ILU (R/B)),
6. PCG - MILU with red/black ordering (MILU (R/B)),
7. PCG - SSOR with red/black ordering (SSOR (R/B)),
8. PCG - m-step Jacobi polynomial preconditioner (different m were experimented with),
9. PCG - polynomial preconditioner with minimum mean square error (m = 2), where $\gamma_0 = 7/6$ and $\gamma_1 = 5/6$ [4] (MMSE, m = 2),
10. PCG - polynomial preconditioner with minimum mean square error (m = 3), where $\gamma_0 = 35/32$, $\gamma_1 = 50/32$, and $\gamma_2 = 35/32$ (MMSE, m = 3),
11. PCG - polynomial preconditioner with minimum mean square error (m = 4), where $\gamma_0 = 37/40$, $\gamma_1 = 49/40$, $\gamma_2 = 91/40$, and $\gamma_3 = 63/40$ (MMSE, m = 4).
12. For the purpose of evaluating the PCG method with the other iterative methods, both the Jacobi and the successive overrelaxation (SOR) method with red/black ordering and with optimal $\omega = \frac{2}{1 + sin(\pi h)}$ was also included in this experiment. Only the results from SOR method is shown in this paper.

5. Results and discussion

5.1 Results

The convergence rates are plotted as a function of N, the number of grid points used, in Figure 7 and 8 (Figure 8 shows the convergence rates of the polynomial preconditioners while Figure 7 shows the convergence rates of the best polynomial preconditioner among those experimented, together with other preconditioners). The convergence rates, expressed in $O(N^{\alpha})$, can be obtained by examining the slope of the corresponding curve and the results are summarized in the following table.

preconditioner	analysis	CM results
CG	$O(N^{0.5})$	$O(N^{0.49})$
ILU (diagonal)	$O(N^{0.5})$	$O(N^{0.45})$
MILU (diagonal)	$O(N^{0.25})$	$O(N^{0.27})$
SSOR (diagonal)	$O(N^{0.25})$	$O(N^{0.27})$
ILU (R/B)	$O(N^{0.5})$	$O(N^{0.49})$
MILU (R/B)	$O(N^{0.5})$	$O(N^{0.50})$
SSOR (R/B)	$O(N^{0.5})$	$O(N^{0.49})$
6-step Jacobi	$O(N^{0.5})$	$O(N^{0.50})$

Table 2 : Theoretical and experimental convergence rates for different preconditioners

The overall execution times for different preconditioners are listed in table 3 below.

preconditioner	64×64	128×128	256×256
SOR (R/B)	4.0 sec	7.7 sec	30.5 sec
CG	7.7 sec	17.1 sec	74.1 sec
ILU (diagonal)	139 sec	530 sec	---
MILU (diagonal)	72 sec	202 sec	---
SSOR (diagonal)	86 sec	245 sec	---
ILU (R/B)	6.1 sec	12.3 sec	51.8 sec
MILU (R/B)	9.4 sec	14.1 sec	66.5 sec
SSOR (R/B)	5.1 sec	10.1 sec	47.3 sec
4-step Jacobi	3.6 sec	7.2 sec	37 sec
6-step Jacobi	3.4 sec	6.6 sec	38 sec
MMSE, m = 2	5.0 sec	9.6 sec	50 sec
MMSE, m = 3	4.2 sec	8.8 sec	43 sec
MMSE, m = 4	3.6 sec	6.9 sec	38 sec

Table 3 : CM execution time for PCG's

The convergence rates of the PCG method using MILU with diagonal ordering as preconditioner for different c is plotted and shown in Figure 9.

The convergence rates of m-step Jacobi preconditioner as a function of the number of terms used for different N is shown in Figure 10 and the corresponding CM execution times are plotted in Figure 11. The convergence rate as well as the CM execution times for SOR (R/B) method are shown in Figure 12 and 13.

5.2 The Connection Machine Statistics

In an attempt to identify bottlenecks in the implementation of the PCG method on the CM, some execution time statistics were gathered on some basic operations of the PCG algorithm. Table 4 shows a breakdown of the execution times for these operations for the polynomial preconditioner with minimum mean square error and m = 4.

	64x64 grid		128x128 grid		256x256 grid	
operation	total time	CM time	total time	CM time	total time	CM time
init	0.05	0.035	0.049	0.035	0.079	0.066
solve	0.063	0.051	0.068	0.051	0.21	0.182
precond	0.026	0.025	0.026	0.025	0.095	0.091
inner	0.012	0.0056	0.012	0.0056	0.024	0.017
mv_prod	0.0075	0.0056	0.0075	0.0056	0.022	0.020
ax+y	0.006	0.003	0.006	0.003	0.014	0.012

Table 4 : CM execution time for different operations in the PCG algorithm

The above table distinguishes between total time and CM time, all in seconds. CM time refers to the time the CM hardware is active, while the total time includes both the front end computer execution time as well as the CM time. The total time varies from time to time on the same operation due to system overhead such as paging. Thus, this total time statistics are taken from the average of 5 - 10 sample runs. The 'solve' includes the operations required for one iteration of the PCG algorithm (including the preconditioning), and excludes the residual calculation. Also 'init' refers to initialization time, 'precond' refers to preconditioning time, 'inner' refers to inner product time, and 'mv_prod' refers to the time to calculate matrix-vector product.

From the table, we can observe that as long as the total number of grid points is less or equal to the number of available processors, the execution time per iteration remains more or less the same. Consequently, the execution time depends primarily on the number of iterations to achieve convergence.

Going from the 128×128 to the 256×256 grid (the CM used in this experiment has 16k processors), the virtual processor ratio (VP ratio) goes from 1 to 4 (now 4 grid points are mapped to 1 physical processor). The data from Table 4 show that the execution times for all operations (including the inner product) go up 2 - 4 times. (The reason that it is not exactly 4 times is probably due to speedup as a result of the pipelining of operations in the CM) Therefore the execution time per iteration is a linear function of the VP ratio, and thus the overall execution time is a function of both the VP ratio and the iteration count.

5.3 Discussion

Table 2 shows that the orders of convergence rates obtained from the experiments confirm those obtained from analysis. From Figure 7 and Table 3, we can see that although the diagonal-ordered preconditioners give much better convergence rates than the red/black-ordered ones, their performance on the CM is not very encouraging (even much worse than the performance of CG method without preconditioning), due to their sequential nature. The execution time of the diagonal-ordered MILU preconditioner, for example, is about 20 times that of the red/black ordered MILU, spending over 90 % of the time in preconditioning. Moreover, data in Table 2 also shows that while the increase in execution time for the other preconditioners is less than a factor of 2 going from 64×64 grid to 128×128 grid, the diagonal-ordered preconditioner takes about 3-4 times as much. From these results, we can conclude that the diagonal-ordered preconditioners are more not suitable for massively parallel machines such as the CM. They are more suitable for machines where the number of processors is much less than the number of grid points.

The red/black ordered preconditioners do effectively utilize the hardware resources on the CM. However, the convergence rates of the red/black ordered preconditioners show only slight improvement (about a factor of 2) over that of the CG method without preconditioning, resulting in execution time improvement of only less than 2.

From Figure 10, it can be observed that with increasing number of terms used in the m-step Jacobi preconditioning, the number of iterations needed to achieve convergence oscillates but continues to decrease slowly. This agrees with the theoretical analysis, as discussed in [8]. However, from Figure 11, it can be seen that the best CM time performance occurs at small number of terms (around 6 for the most cases). So there is a point at which the improvement of convergence rate is counter-balanced by the increase in the amount of work for preconditioning.

The convergence rate plot in Figure 7 and 8 shows that polynomial preconditioning is very competitive among all other preconditioners. The minimum mean square error preconditioner (MMSE) shows only slight improvement over the m-step Jacobi preconditioner when the same number of terms is used. The execution time improvement should be more pronounced if the floating point hardware is used since the time for multiply by the coefficients is reduced. Overall, it seems worthwhile to search for better coefficients to achieve better convergence rate for this class of preconditioner.

The CM performance of the SOR (R/B) method shows that it is still one of the most efficient methods on massively parallel machines. However, this excellent performance is due to the use of optimal parameter, which, in a lot of applications, are difficult to obtained. It is shown from Figure 12 and 13 that the convergence rate is very sensitive to the parameter used, and slight deviation from the optimal parameter can result in poorer convergence rate and thus longer execution time. Moreover, even with the use of optimal parameter for SOR (R/B), the execution time performance is still worse than the 6-step Jacobi PCG method. However, even though the execution time performance of the SOR (R/B) is worse than that of the 6-step Jacobi PCG for the 128×128 grid, it is better for the 256×256 grid. The reason is that for the 128×128 grid, the VP ratio is 1, meaning that the each black and red point is mapped to a different physical processor. Since each sweep involves only either all black points or all red points, the machine utilization is at most 50 %. However, when the VP ratio goes up to 4, 2 black points and 2 red points are mapped to the same physical processor, and we can observe that all physical processors are active during sweeping, updating either the black points or the red points. Thus, the processor utilization is increased to close to 100 %, and the total execution time is reduced by a factor of 2.

The conclusions that can be drawn from the result are first that fast convergence rate offered by a preconditioner does not guarantee lowest execution time on parallel machines (degree of parallelism in the preconditioner also has to be taken into consideration); and that parallel preconditioners such as the R/B SSOR only improves the performance by a small amount (due to its slight improvement in convergence rate). Thus, the best preconditioner for parallel machines should strike a balance between total parallelism and fastest convergence rate.

6. Conclusion

The conclusion from this CM experiments is that to search for a better preconditioner on parallel machines, tradeoffs have to be made between fast convergence rate and the degree of parallelism in the preconditioner, as these two are mutually conflicting on execution time performance on these machines. A preconditioner showing promises of better performance in both the convergence rate and the degree of parallelism is the hierarchical basis preconditioning [10]. Its analysis and performance on the CM is currently under study.

Acknowledgement

The author would especially like to thank Professor Tony Chan for his many suggestions, advices, and encouragement leading to the completion of this paper. Also many thanks is to Professor Jay Kuo for his advices and support. The author would also like to thank the Parallel Processing Group at the Information Sciences Institute (ISI) of the University of Southern California (USC) for providing the CM computing facilities as well as instructions of how to use them prior to the installation of CM2 at UCLA.

References :

1. Chan, Tony F., Kuo, C. C. Jay, and Tong, Charles, "Parallel Elliptic Preconditioners : Fourier Analysis and Performance on the Connection Machine", Department of Mathematics, CAM report 88-22, UCLA, August 1988.

2. Kuo, C.-C. Jay, and Chan, Tony F., "Two-color Fourier Analysis of Iterative Algorithms for Elliptic Problems with Red/Black Ordering", CAM Report 88-15, Department of Mathematics, UCLA, 1988.

3. Chan, Tony F., and Elman, Howard C., "Fourier Analysis of Iterative Methods for Elliptic Problems", CAM Report 87-04, Department of Mathematics, UCLA, 1988. To appear in *SIAM Review.*

4. Johnson, O., Micchelli, C. A., Paul, G., "Polynomial Preconditioners for Conjugate Gradient Calculations", *SIAM J. Numer. Anal.*, Vol. 20, No. 2, April 1983, pp. 362-376.

5. McBryan, O. A., "State-of-the-art in Highly Parallel Computer Systems, " in *Parallel Computations and Their Impact Mechanics*, ed. A. K. Noor, pp. 31-46, The American Society of Mechanical Engineers, New York, 1987.

6. Donato, J., "Fourier Analysis of Polynomial Preconditioners for the 5-point Laplacian," Term Paper, Department of Mathematics, UCLA, 1988.

7. Ortega, J. M., and Voigt, R. G., "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review*, Vol. 27, No. 2, pp. 149-240, June 1985.

8. Ortega, J. M., *Introduction to Parallel and Vector Solution of Linear Systems*, Frontier of Computer Science, Plenum Press, New York, 1988, Section 3.4.

9. *Notes on Introductory Course in Programming the Connection Machine System*, offered by the Thinking Machine Corporation.

10. Adam, L. M., and Ong, E. G., "A Comparison of Preconditioners for GMRES on Parallel Computers," in *Parallel Computations and Their Impact on Mechanics*, ed. A. K. Noor, pp. 171-186, The American Society of Mechanical Engineers, New York, N. Y., 1987.

11. Golub, G. H., Van Loan, C. F.,*Matrix Computations*, the Johns Hopkin University Press, 1983, chapter 6.

12. Dupont, T., Kendall, R. P., and Rachford, H. H. Jr., "An Approximate Factorization Procedure for Solving Self-adjoint Difference Equations," *SIAM J. Numer. Anal.*, vol. 5, No. 3, pp. 559-573.

13. Meijerink, J. A., and Van der Vorst, H. A., "An iterative Solution Method for Linear Systems of which the Coefficient matrix is a symmetric M-Matrix," *Math. Comp.*, Vol. 31, no. 137, pp. 148-162, 1977.

14. Axelsson, O., "A Generalized SSOR Method," *BIT*, Vol. 13, pp. 443-467, 1972.

15. Chandra, R., *Conjugate Gradient Methods for Partial Differential Equations*, Ph. D. Thesis, Computer Science Department, Yale University, 1978.

$$\begin{array}{ccc} & -\frac{1}{4} & \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ & -\frac{1}{4} & \end{array}$$

Figure 1 : stencil representation of $A_{j,k}$ operator

6	7	8	9	10	11
5	6	7	8	9	10
4	5	6	7	8	9
3	4	5	6	7	8
2	3	4	5	6	7
1	2	3	4	5	6

(a)

2	1	2	1	2	1
1	2	1	2	1	2
2	1	2	1	2	1
1	2	1	2	1	2
2	1	2	1	2	1
1	2	1	2	1	2

(b)

Figure 2 : (a) diagonal and (b) parallel red/black orderings for 6×6 grid

$$\begin{array}{cc} -\frac{1}{4} & \frac{a}{4} \\ & -\frac{1}{4} \end{array} \qquad \begin{array}{cc} -\frac{1}{a} & \\ 1 & -\frac{1}{a} \end{array} \qquad \begin{array}{ccc} \frac{1}{4a} & -\frac{1}{4} & \\ -\frac{1}{4} & \frac{a}{4}+\frac{1}{2a} & -\frac{1}{4} \\ & -\frac{1}{4} & \frac{1}{4a} \end{array}$$

$$L_{j,k} \qquad\qquad U_{j,k} \qquad\qquad M_{I\,(j,k)}$$

$$L_{j,k} = \frac{a}{4} - \frac{1}{4} E_x^{-1} - \frac{1}{4} E_y^{-1}$$

$$U_{j,k} = 1 - \frac{1}{a} E_x - \frac{1}{a} E_{y,}$$

$$M_{I\,(j,k)} = \frac{a}{4} + \frac{1}{2a} - \frac{1}{4} (E_x + E_y + E_x^{-1} + E_y^{-1}) + \frac{1}{4a} (E_x E_y^{-1} + E_x^{-1} E_y)$$

Figure 3 : Stencil representations of local operators for the ILU preconditioner

(a) diagonal ordering

$(j,\ k)$ *red* :

$$\begin{array}{ccc} & & \\ & 1 & \\ & & \\ & L_{j,k} & \end{array} \qquad \begin{array}{ccc} & -\frac{1}{4} & \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ & -\frac{1}{4} & \\ & U_{j,k} & \end{array} \qquad \begin{array}{ccc} & -\frac{1}{4} & \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ & -\frac{1}{4} & \\ & M_{lr\,(j,k)} & \end{array}$$

$(j,\ k)$ *black* :

$$\begin{array}{ccccc} & & \frac{1}{16} & & \\ & \frac{1}{8} & -\frac{1}{4} & \frac{1}{8} & \\ \frac{1}{16} & -\frac{1}{4} & 1 & -\frac{1}{4} & \frac{1}{16} \\ & \frac{1}{8} & -\frac{1}{4} & \frac{1}{8} & \\ & & \frac{1}{16} & & \\ & & M_{lb\,(j,k)} & & \end{array}$$

$$L_{j,k} = \begin{cases} 1\,, & (j,k)\ red \\ 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)\,, & (j,k)\ black \end{cases}$$

$$U_{j,k} = \begin{cases} 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)\,, & (j,k)\ red \\ \frac{3}{4}\,, & (j,k)\ black \end{cases}$$

$$M_{lrb\,(j,k)} = \begin{cases} 1 - \frac{1}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,)\,, & (j,k)\ red \\ \frac{3}{4} - \frac{1}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,) + \frac{1}{16}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)^2\,, & (j,k)\ black \end{cases}$$

Figure 3: Stencil representations of local operators for the ILU preconditioner

(b) red/black ordering

$(j,\ k)$ *red* :

$$
\begin{array}{c}
1+\delta \\[2ex]
L_{j,k}
\end{array}
\qquad\qquad
\begin{array}{ccc}
 & \frac{-1}{4(1+\delta)} & \\
\frac{-1}{4(1+\delta)} & 1 & \frac{-1}{4(1+\delta)} \\
 & \frac{-1}{4(1+\delta)} & \\[2ex]
 & U_{j,k} &
\end{array}
\qquad\qquad
\begin{array}{ccc}
 & -\frac{1}{4} & \\
-\frac{1}{4} & 1+\delta & -\frac{1}{4} \\
 & -\frac{1}{4} & \\[2ex]
 & M_{Mr\,(j,k)} &
\end{array}
$$

$(j,\ k)$ *black* :

$$
\begin{array}{ccc}
 & -\frac{1}{4} & \\
-\frac{1}{4} & \frac{\delta^2+2\delta}{1+\delta} & -\frac{1}{4} \\
 & -\frac{1}{4} & \\[2ex]
 & L_{j,k} &
\end{array}
\qquad\qquad
\begin{array}{c}
1 \\[2ex]
U_{j,k}
\end{array}
\qquad\qquad
\begin{array}{ccccc}
 & & \frac{1}{16(1+\delta)} & & \\
 & \frac{1}{8(1+\delta)} & -\frac{1}{4} & \frac{1}{8(1+\delta)} & \\
\frac{1}{16(1+\delta)} & -\frac{1}{4} & \frac{\delta^2+2\delta+\frac{1}{4}}{1+\delta} & -\frac{1}{4} & \frac{1}{16(1+\delta)} \\
 & \frac{1}{8(1+\delta)} & -\frac{1}{4} & \frac{1}{8(1+\delta)} & \\
 & & \frac{1}{16(1+\delta)} & & \\[2ex]
 & & M_{Mb\,(j,k)} & &
\end{array}
$$

$$
L_{j,k} = \begin{cases} 1+\delta, & (j,k)\ \textit{red} \\ 1+\delta - \dfrac{1}{1+\delta} - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ \textit{black} \end{cases}
\qquad
U_{j,k} = \begin{cases} 1 - \dfrac{1}{4\,(1+\delta)}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ \textit{red} \\ 1, & (j,k)\ \textit{black} \end{cases}
$$

$$
M_{Mrb\,(j,k)} = \begin{cases} 1+\delta - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k)\ \textit{red} \\ 1+\delta - \dfrac{1}{1+\delta} - \dfrac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}) + \dfrac{1}{16\,(1+\delta)}\,(E_x + E_x^{-1} + E_y + E_y^{-1})^2, & (j,k)\ \textit{black} \end{cases}
$$

Figure 4: Stencil representations of local operators for the MILU preconditioner with parallel red/black ordering

$$\begin{array}{ccc} & & \\ -\frac{\omega}{4} & 1 & \\ & -\frac{\omega}{4} & \end{array} \qquad \begin{array}{ccc} & -\frac{\omega}{4} & \\ & 1 & -\frac{\omega}{4} \\ & & \end{array} \qquad \begin{array}{ccc} \frac{\omega^2}{16} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1+\frac{\omega^2}{8} & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \frac{\omega^2}{16} \end{array}$$

$$D_{j,k} - \omega L_{j,k} \qquad\qquad D_{j,k} - \omega L^T_{j,k} \qquad\qquad M_{S\,(j,k)}$$

$$D_{j,k} = 1\,, \quad L_{j,k} = \frac{1}{4}\,(E_x^{-1} + E_y^{-1})\,, \quad L^T_{j,k} = \frac{1}{4}\,(E_x + E_y)\,.$$

$$D_{j,k} - \omega L_{j,k} = 1 - \frac{\omega}{4}\,(E_x^{-1} + E_y^{-1})$$

$$D_{j,k} - \omega L^T_{j,k} = 1 - \frac{\omega}{4}\,(E_x + E_y)$$

$$M_{S\,(j,k)} = 1 - \frac{\omega}{4}\,(E_x + E_y + E_x^{-1} + E_y^{-1}) + \frac{\omega^2}{16}\,(2 + E_x^{-1} E_y + E_x E_y^{-1})\,.$$

Figure 5 : Stencil representations of local operators for the SSOR preconditioner

(a) diagonal ordering

$(j,\ k)$ *red* :

$$\begin{array}{c} 1 \\ \\ D_{j,k} - \omega L_{j,k} \end{array} \qquad \begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \\ \\ \multicolumn{3}{c}{D_{j,k} - \omega L_{j,k}^T} \end{array} \qquad \begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \\ \\ \multicolumn{3}{c}{M_{Sr\,(j,k)}} \end{array}$$

$(j,\ k)$ *black* :

$$\begin{array}{ccc} & -\frac{\omega}{4} & \\ -\frac{\omega}{4} & 1 & -\frac{\omega}{4} \\ & -\frac{\omega}{4} & \\ \\ \multicolumn{3}{c}{D_{j,k} - \omega L_{j,k}} \end{array} \qquad \begin{array}{c} 1 \\ \\ D_{j,k} - \omega L_{j,k}^T \end{array} \qquad \begin{array}{ccccc} & & \frac{\omega^2}{16} & & \\ & \frac{\omega^2}{8} & -\frac{\omega}{4} & \frac{\omega^2}{8} & \\ \frac{\omega^2}{16} & -\frac{\omega}{4} & 1+\frac{\omega^2}{4} & -\frac{\omega}{4} & \frac{\omega^2}{16} \\ & \frac{\omega^2}{8} & -\frac{\omega}{4} & \frac{\omega^2}{8} & \\ & & \frac{\omega^2}{16} & & \\ \\ \multicolumn{5}{c}{M_{Sb\,(j,k)}} \end{array}$$

$$D_{j,k} = 1\ ,$$

$$L_{j,k} = \begin{cases} 0\ , & (j,k)\ \textit{red} \\ 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)\ , & (j,k)\ \textit{black} \end{cases} \qquad U_{j,k} = \begin{cases} 1 - \frac{1}{4}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)\ , & (j,k)\ \textit{red} \\ 0\ , & (j,k)\ \textit{black} \end{cases}$$

$$M_{Srb\,(j,k)} = \begin{cases} 1 - \frac{\omega}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,)\ , & (j,k)\ \textit{red} \\ 1 - \frac{\omega}{4}\,(\,E_x + E_x^{-1} + E_y + E_y^{-1}\,) + \frac{\omega^2}{16}\,(E_x + E_x^{-1} + E_y + E_y^{-1}\,)^2\ , & (j,k)\ \textit{black} \end{cases}$$

Figure 5 : Stencil representations of local operators for the SSOR preconditioner

(b) red/black ordering

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 3 & 4 & 3 & & \\
 & 3 & 8 & 25 & 8 & 3 & \\
1 & 4 & 25 & 80 & 25 & 4 & 1 \\
 & 3 & 8 & 25 & 8 & 3 & \\
 & & 3 & 4 & 3 & & \\
 & & & 1 & & &
\end{array}$$

$$64 \times M_{P,3}^{-1}$$

Figure 6 : Stencil representation of the local operator for the polynomial preconditioner - 4-step Jacobi

Figure 7 - Convergence Rates for SOR (R/B) & PCG Methods

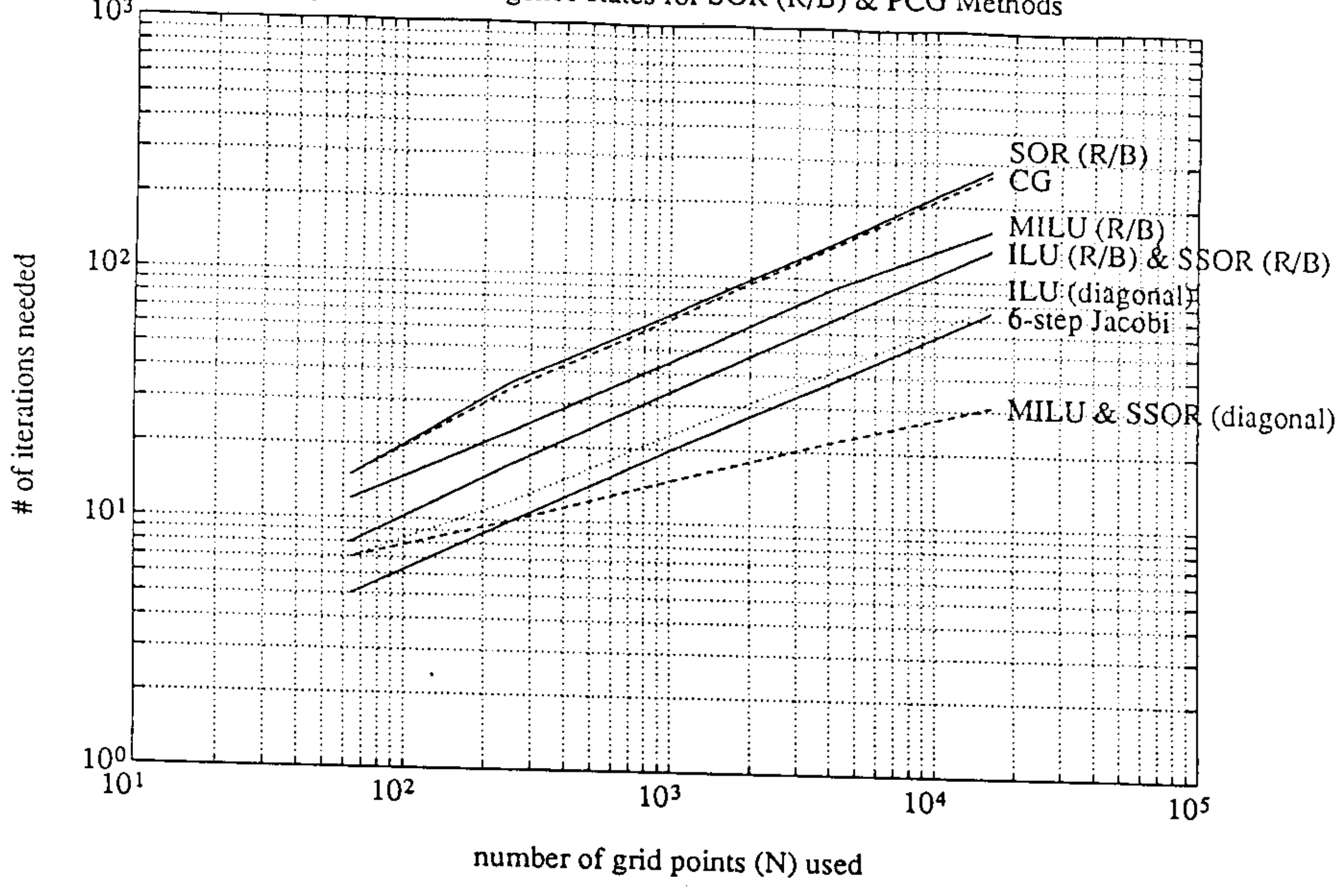


Figure 8 - Convergence Rates for different Polynomial Preconditioners

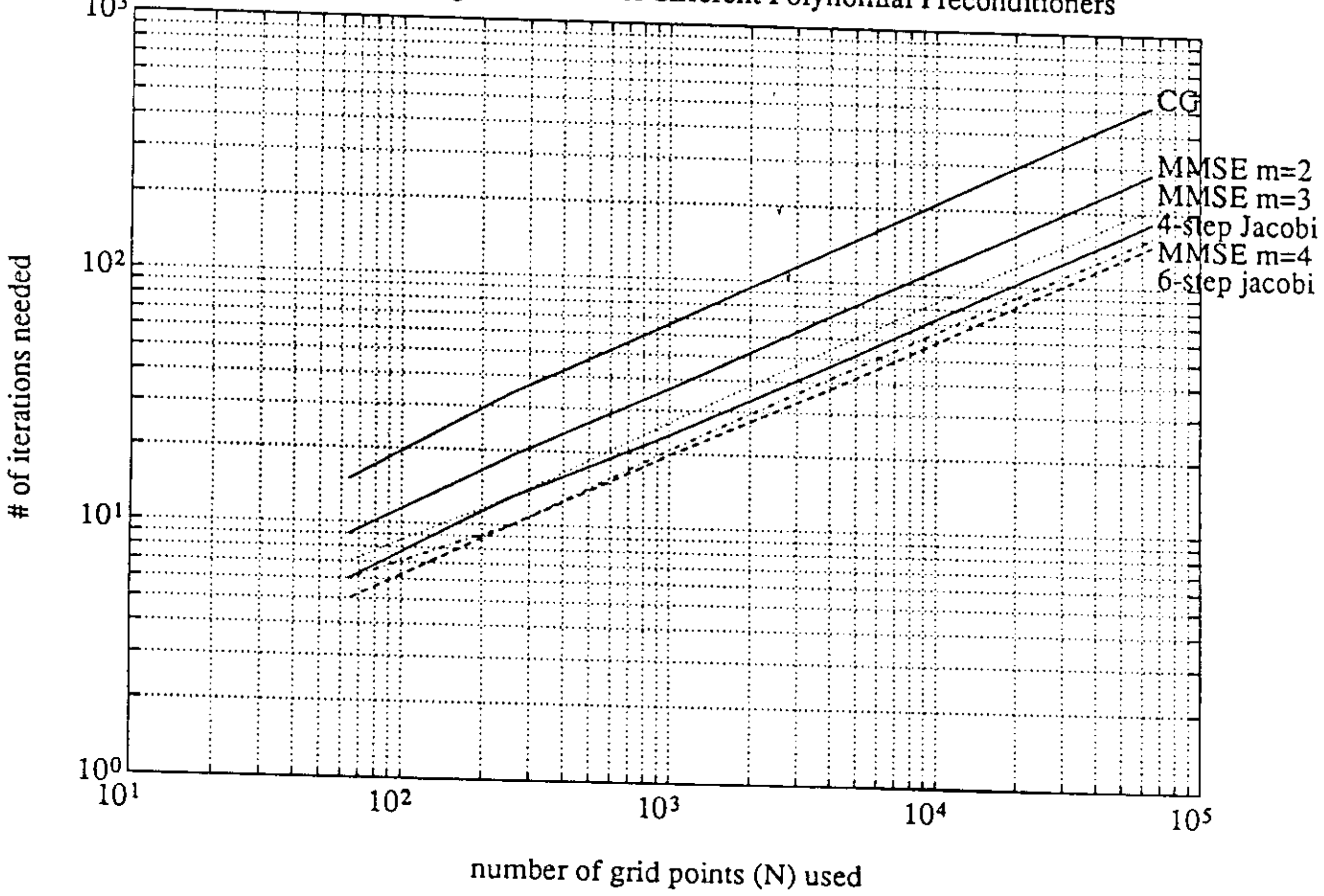


Figure 9 - MILU Convergence Rates vs. value of c

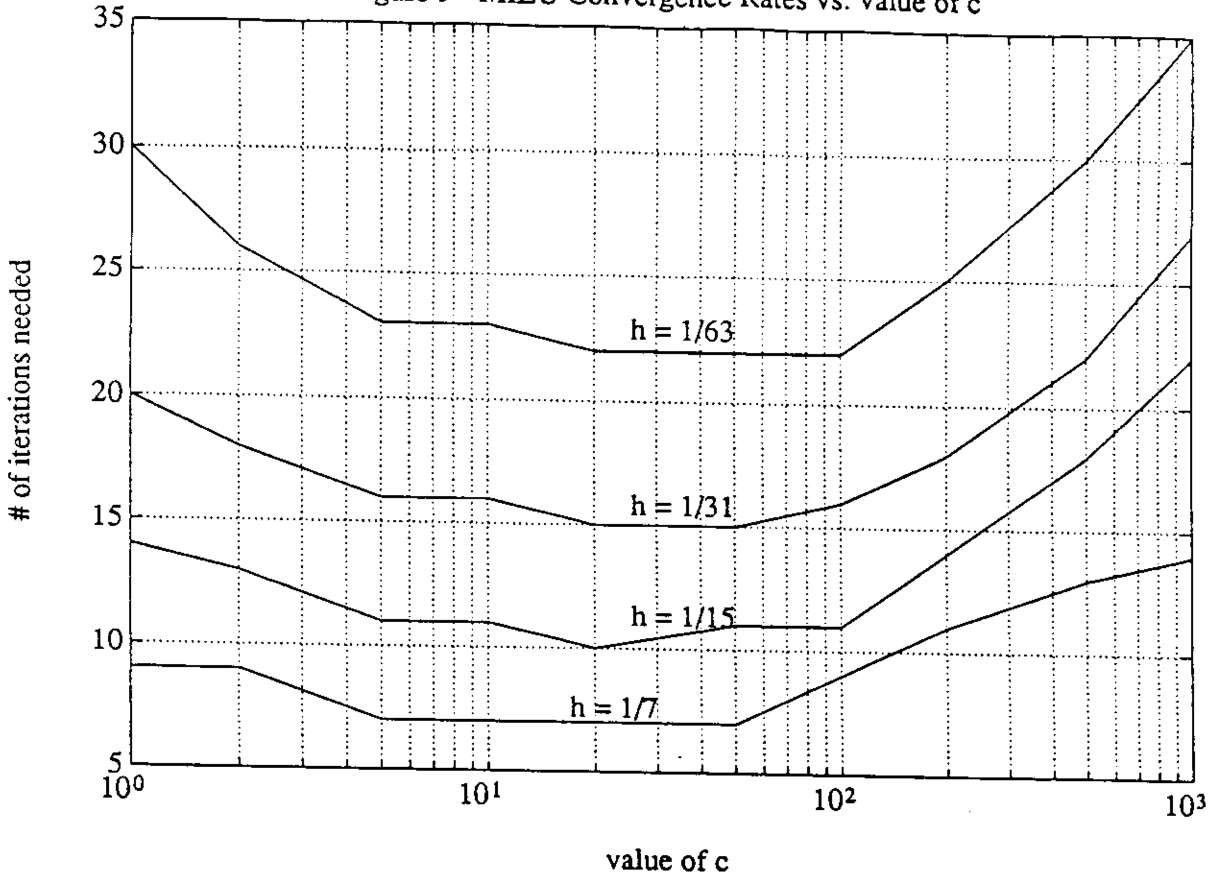


Figure 10 - m-step Jacobi Convergence Rates vs. m

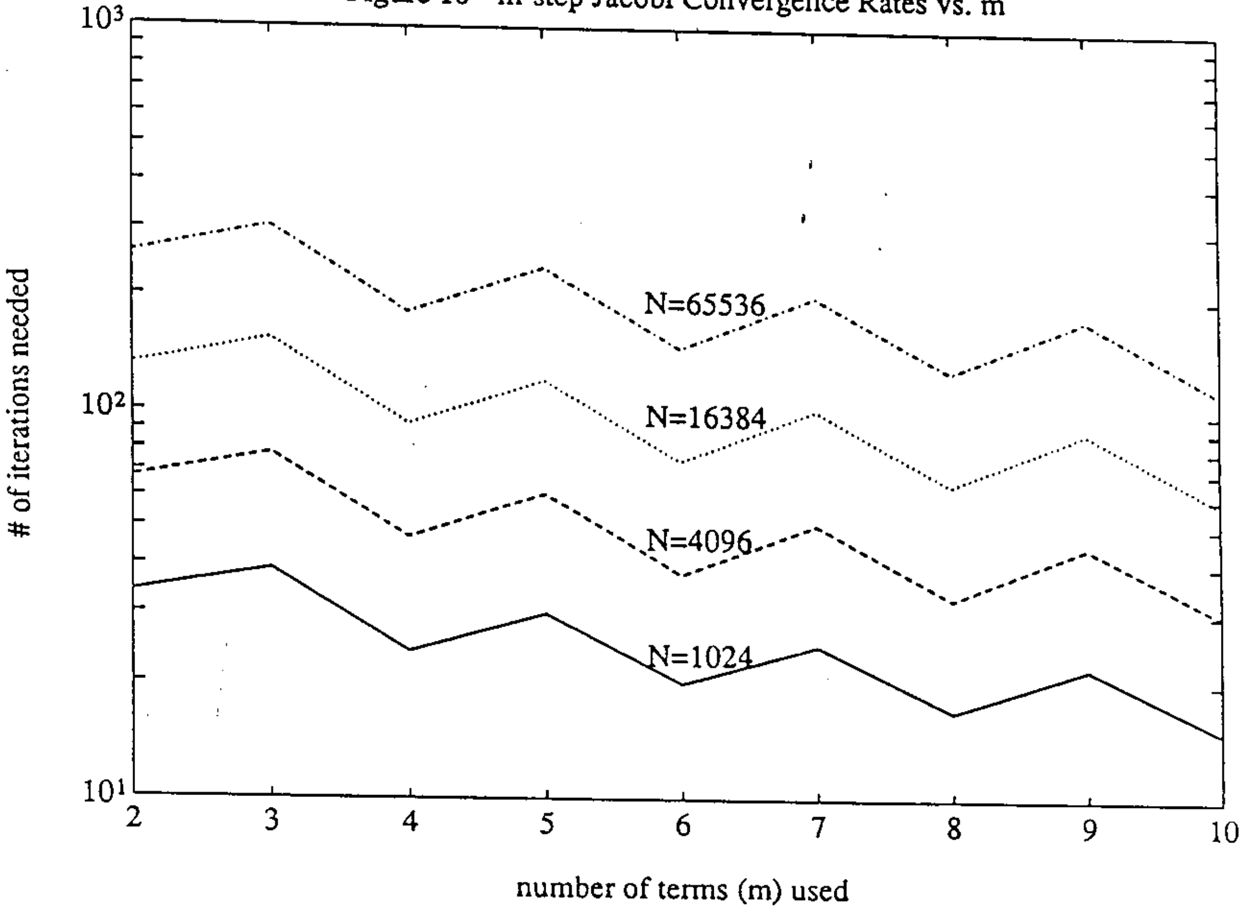
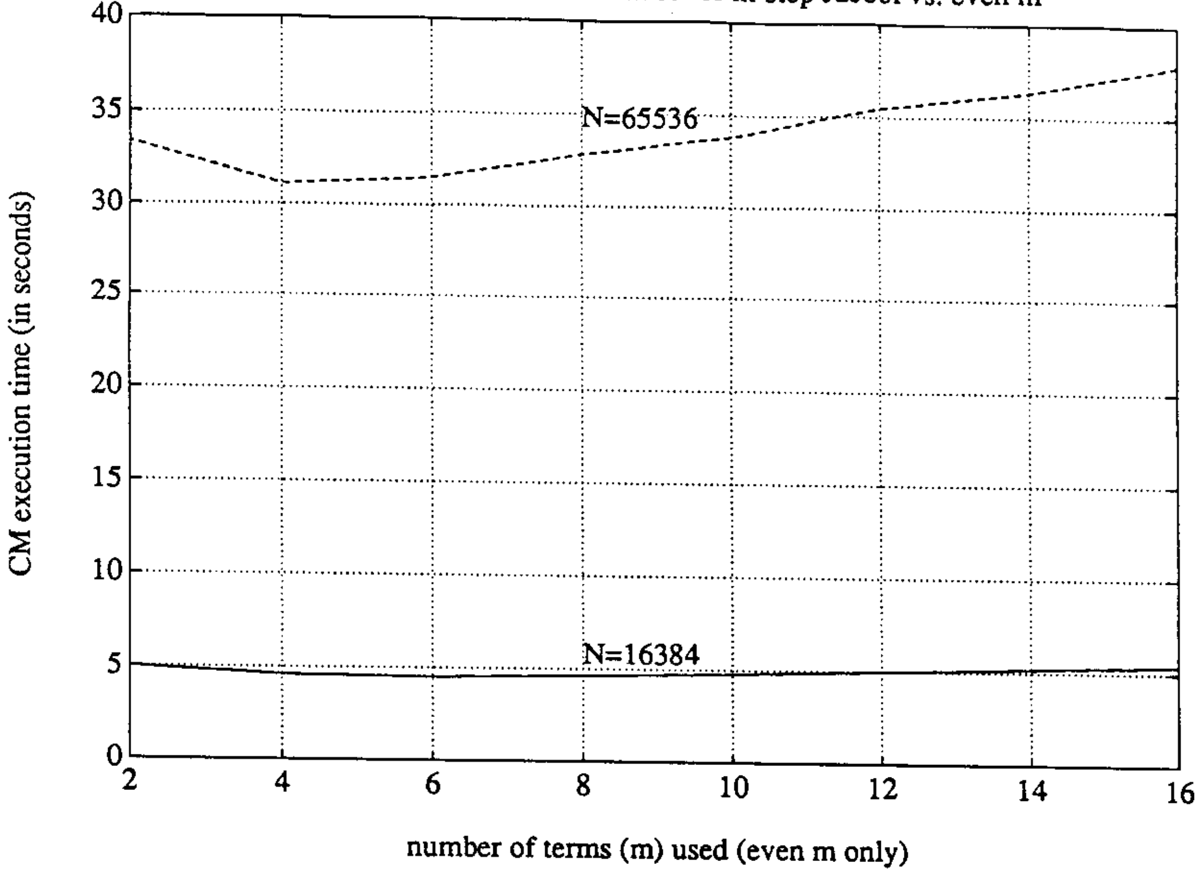
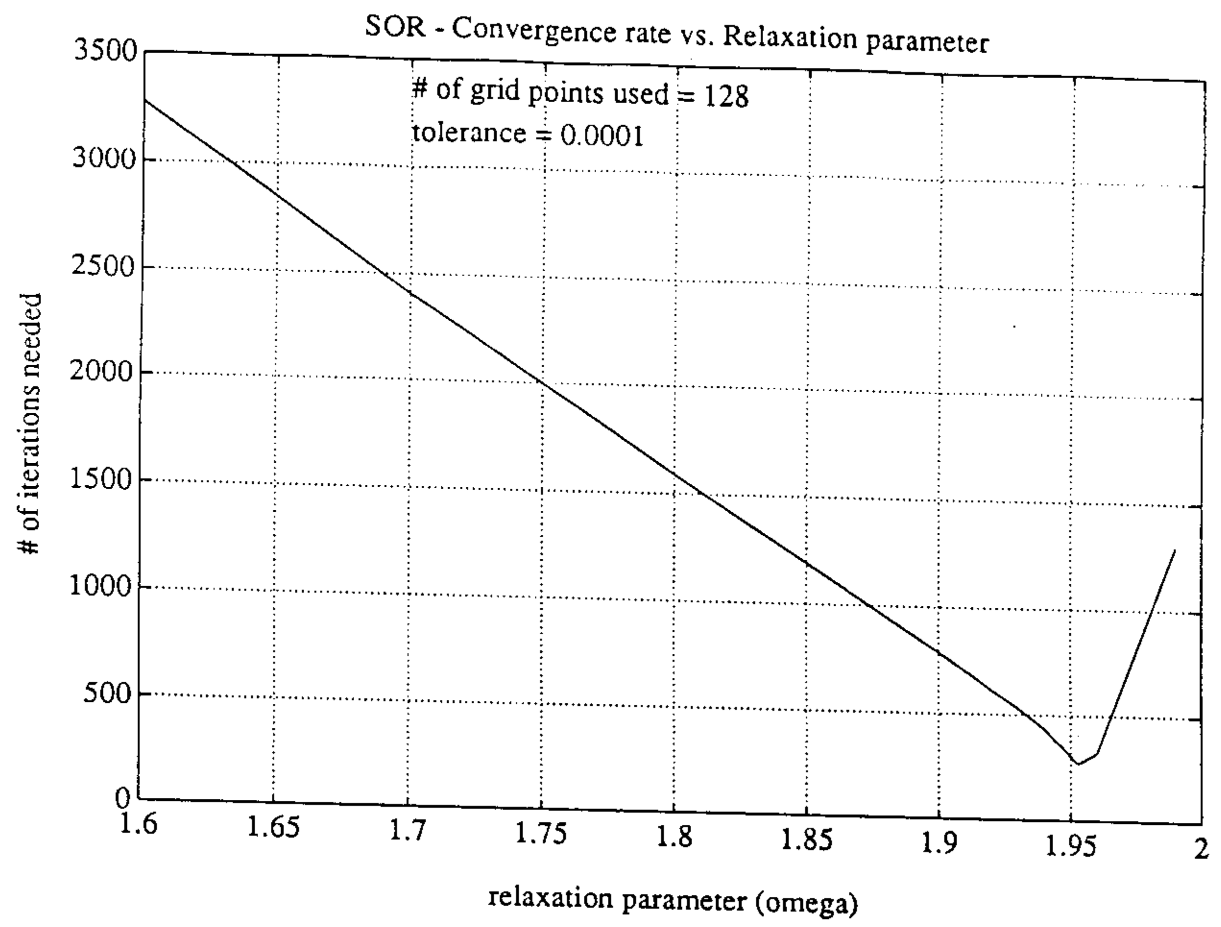
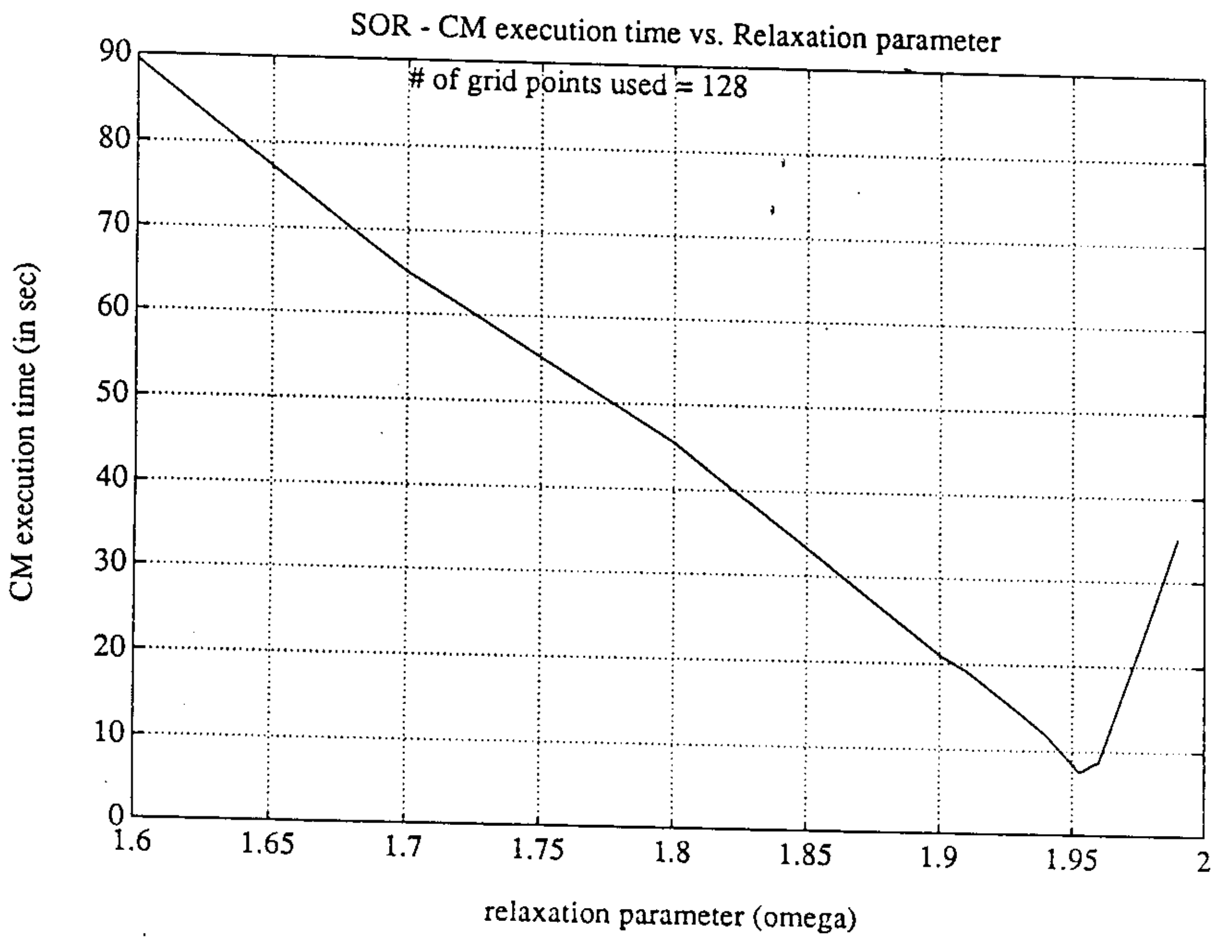


Figure 11 - CM Execution Times for m-step Jacobi vs. even m




SOR - Convergence rate vs. Relaxation parameter
# of grid points used = 128
tolerance = 0.0001
# of iterations needed
relaxation parameter (omega)
3500
3000
2500
2000
1500
1000
500
0
1.6
1.65
1.7
1.75
1.8
1.85
1.9
1.95
2


SOR - CM execution time vs. Relaxation parameter
# of grid points used = 128
CM execution time (in sec)
relaxation parameter (omega)
90
80
70
60
50
40
30
20
10
0
1.6
1.65
1.7
1.75
1.8
1.85
1.9
1.95
2