Efficient Implementation of the Multigrid Preconditioned Conjugate Gradient Method on Distributed Memory Machines

Osamu Tatebe*† Yoshio Oyanagi†

Department of Information Science
the University of Tokyo
Bunkyo-ku, Tokyo 113 JAPAN

Abstract

A multigrid preconditioned conjugate gradient (MGCG) method[15], which uses the multigrid method as a preconditioner for the CG method, has a good convergence rate even for the problems on which the standard multigrid method does not converge efficiently. This paper considers a parallelization of the MGCG method and proposes an efficient parallel MGCG method on distributed memory machines. For the good convergence rate of the MGCG method, several difficulties in parallelizing the multigrid method are successfully settled. Besides, the parallel MGCG method on Fujitsu multicomputer AP1000[8] has high performance and it is more than 10 times faster than the Scaled CG (SCG) method[6].

1 Introduction

Parallelization of the multigrid method has been studied and several parallel multigrid methods have been implemented. One natural parallelization approach is governed by the grid partitioning principles. Sbosny[13] analyzed and implemented parallel multigrid method using the domain decomposition ideas. This algorithm is mathematically equal to the sequential multigrid algorithm. Besides the domain decomposition, the standard multigrid method with a highly parallel smoother is easy to be parallelized. However, the convergence rate of this method degenerates on Poisson's problems with severe coefficient jumps. For efficient convergence on that equation, it has been reported that line (or plane) relaxation and semicoarsening are effective[1, 3]. Unfortunately, since these techniques have poor and awkward parallelism, they are inefficient on distributed memory machines. Dendy[4] implemented semicoarsening multigrid algorithm suitable for SIMD machines on the CM-2. This method requires half amount of relaxation computation as that in the full coarsening multigrid case. However this method requires line or plane relaxation. An alternative way of achieving robustness is to use multiple coarse grids. Mulder[10] and Naik[11] proposed multiple semicoarsed grid (MSG) algorithm and Overman[12] implemented this algorithm on distributed memory machines. This method has ample parallelism and relative robustness. However, the MSG algorithm needs a larger amount of computation than the ordinary multigrid method.

The MGCG method[15] is a PCG method that uses the multigrid method as a preconditioner. In [15], we showed the MGCG method has robustness on Poisson's problems with severe coefficient jumps even if it uses highly parallel point relaxation. This paper investigates an efficient parallelization of the MGCG method on distributed memory machines. Besides, the most efficient parallel MGCG method is evaluated in terms of parallel efficiency on an AP1000. This paper is organized as follows. Section 2 explains the MGCG method briefly. Section 3 discusses the parallelization of the MGCG method on distributed memory machines. In Section 4, we implement the MGCG method on the multicomputer AP1000, and evaluate it in terms of parallel efficiency. Finally, the MGCG method is compared with the SCG method[6]. The main conclusions are summarized in Section 5.

2 The MGCG Method

The MGCG method is the PCG method that uses the multigrid method as a preconditioner. When a target linear equation is $L_l \boldsymbol{x} = \boldsymbol{f}$, iteration of the

*JSPS Research Fellow
†E-mail: {tatebe,oyanagi}@is.s.u-tokyo.ac.jp

MGCG method is described by Figure 1. Let an initially approximate vector be x^0 and set an initial residual $r^0 = f - L_l x^0$. Next $L_l \tilde{r}^0 = r^0$ is approximately solved by the multigrid method and set an initial direction vector be $p^0 = \tilde{r}^0$. Then loop of Figure 1 is iterated until convergence.

```
$i = 0$;
while ( !convergence ) {
  $\alpha_i = (\tilde{r}_i, r_i)/(p_i, L_l p_i)$;
  $x_{i+1} = x_i + \alpha_i p_i$;
  $r_{i+1} = r_i - \alpha_i L_l p_i$;
  convergence test;
  Relax $L_l \tilde{r}_{i+1} = r_{i+1}$ using the Multigrid
method ———————— (A)
  $\beta_i = (\tilde{r}_{i+1}, r_{i+1})/(\tilde{r}_i, r_i)$;
  $p_{i+1} = \tilde{r}_{i+1} + \beta_i p_i$;
  $i$++;
}
```

Figure 1: Iteration of the MGCG method

```
Vector MG($L_l$, $f$, $x$, $\gamma$, $\mu_1$, $\mu_2$)
{
  if ($l$ == coarsest_level) Solve $L_l x = f$;
  else {
    $x$ = pre_smoothing($L_l$, $f$, $x$, $\mu_1$);
    $d$ = restrict($f - L_l x$);
    $\nu$ = initial_$x$;
    repeat ($\gamma$) $\nu$ = MG($L_{l-1}$, $d$, $\nu$, $\gamma$, $\mu_1$, $\mu_2$);
    $x = x$ + prolongate($\nu$);
    $x$ = post_smoothing($L_l$, $f$, $x$, $\mu_2$);
  }
  return $x$;
}
```

Figure 2: The multigrid method

In Figure 1, (A) is the part of a multigrid preconditioning. Figure 2 explains the multigrid method. This function is recursively defined, where a sequence of coefficient matrices is $\{L_i\}$ $(0 \le i \le l)$. μ_1 and μ_2 are the number of pre- and post-smoothing iterations respectively. Multigrid cycle depends on γ, μ_1 and μ_2 of Figure 2. When $\gamma = 1$ and $\mu_1 = \mu_2 \neq 0$, it is called V-cycle multigrid method. When $\gamma = 2$ and $\mu_1 = \mu_2 \neq 0$, it is called W-cycle multigrid method. When $\gamma = 1$ and $\mu_1 = 0$, $\mu_2 \neq 0$, it is called the *sawtooth cycle*. These cycles are depicted by Figure 3. Popular multigrid cycles are V-cycle of $\gamma = 1$ and $\mu_1 = \mu_2 = 1$ and W-cycle of $\gamma = 2$ and $\mu_1 = \mu_2 = 2$.

A transfer operation of vectors on a grid level i to vectors on a grid level $i-1$ is called *restriction*, and an inverse operator is called *prolongation*. The matrices

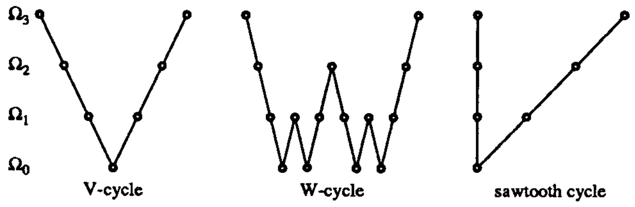

Figure 3: Multigrid cycle

that represent the operations of restriction and prolongation are written as r and p respectively in this paper. One of the present authors[15] proved that the multigrid method is a mathematically valid preconditioner of the PCG method if the following relation:

$$r = b p^T, \quad \mu_1 = \mu_2 \neq 0, \tag{1}$$

where b is a scalar constant, is satisfied and if the pre- and post-smoothings are identical and symmetric methods. Some examples of symmetric smoothing are the Red-Black symmetric SOR (RB-SSOR) smoothing, the multi-color SSOR smoothing and the ADI smoothing. An example of prolongation is a standard transfer operation interpolated bilinearly. The prolongation and its adjoint restriction are symbolized by the following stencils:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix},$$

respectively. We refer to the MGCG method with μ iterations of the RB-SSOR smoother as MGCG(RB, γ, μ, l) method, where l is the number of grid levels.

3 Parallelization of the MGCG Method

This section considers a parallelization of the MGCG method on distributed memory machines. For the CG part, efficient vectorization or parallelization is straightforward, since one iteration of the CG method consists of three inner products, one matrix-vector multiplication, and three vector additions. Therefore we investigate the parallelization of the multigrid method mainly in this section.

3.1 Difficulties of parallelization of the multigrid method

When the multigrid method is parallelized on distributed memory machines, the following difficulties arise.

1. (Algorithmic problem) For robustness, the multigrid method should use line (or plane) relaxation and (or) semicoarsening.
2. (Implementation problem) In comparatively coarse grids with the number of processors,
    (a) communication latency is not hidden.
    (b) usage rate of processors is low.

With a highly parallel smoothing, like point relaxation, the multigrid method has a poor convergence rate on Poisson's problem with severe coefficient jumps. If the multigrid method uses a robust smoother, for example alternative line (or plane) relaxation, on distributed memory machines, it has been reported that parallel efficiency of the multigrid method gains about 60% and this parallel efficiency is a very satisfactory result[7]. However, the parallel efficiency of highly parallel smoother achieves more than 100%. This parallel efficiency is quite attractive.

The number of communication data is $O(\sqrt{N})$ and computational complexity is $O(N)$ in two-dimensional case, where N is the problem size. Thus, on finer grid, the communication latency can be hidden by overlapping communication with computation, however, on comparatively coarse grid, communication latency is not hidden any more. That is, the communication overhead is critical on coarser grid.

3.2 Parallelization of the MGCG method

In the parallelization of the multigrid method, there are the difficulties described in the previous subsection. However in the MGCG method, there is no algorithmic problem. That is because the MGCG method has relative robustness even if it uses highly parallel smoother. The rest is the implementation problem. The problem is considered as a trade-off between the convergence rate of the MGCG method and parallel efficiency of target architectures. If a linear equation in the coarse grid correction is not solved exactly on the coarsest grid, the convergence rate of the multigrid method depends on the mesh size of the coarsest grid as robust as the multigrid method is. On the other hand, highly parallel efficiency is achieved when each processor has plenty of data on the coarsest grid since the number of communication data is relatively small. In the following subsections, the average convergence rate of the MGCG method with fewer number of grid levels is evaluated by some numerical experiments. Also, the eigenvalue distribution of the multigrid preconditioned matrix is analyzed.

3.3 Numerical experiments

In parallel implementation, communication overhead and load imbalance are crucial to performance on coarse grids compared with the number of processors. Therefore we investigate convergence behavior of the MGCG method with a smaller number of grid levels by several numerical examples.

3.3.1 Target problems

Two-dimensional Poisson's equation with Dirichlet boundary condition:

$$-\nabla (k\nabla u) = f \quad \text{in} \quad \Omega = [0,1] \times [0,1]$$

$$\text{with} \quad u = g \quad \text{on} \quad \partial\Omega,$$

where k is a real function, is considered. In the numerical experiments, the following two kinds of Dirichlet conditions g, diffusion coefficient k and source terms f are set up.

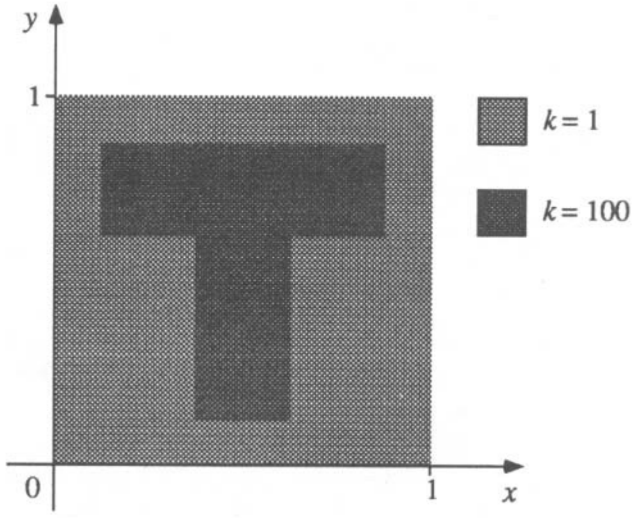

Figure 4: the diffusion coefficients of problem 2

Problem 1 Diffusion coefficient is uniform, and source term is equal to 0. Boundary condition is $g = 0$ except $y = 1$ and $g = 3x(1-x)$ on $y = 1$.

Problem 2 Diffusion coefficient is depicted by Figure 4. Source term is $f = 80$ in $0 \leq x,\ y \leq 0.5$ and $0.5 \leq x,\ y \leq 1$, and $f = -80$ in the other region. Boundary condition g is always equal to 0.

Problem 1 is a simple Poisson's problem. This case has been studied theoretically and reported the standard multigrid method is quite efficient[14]. Problem 2 is a case with severe coefficient jumps, therefore the problem matrix has a rich eigenvalue distribution. It has been widely known the convergence rate of the standard multigrid method degenerates on this problem.

These problems are discretized to 256×256 meshes by the finite element method. These coefficient matrices become symmetric, positive definite and block tridiagonal.

Table 1: Average convergence rate

# grids	3	4	5	6	7
problem 1					
# iter.	59	30	16	9	7
conv. rate	0.73	0.54	0.29	0.13	0.070
time (sec.)	117.9	60.9	32.8	18.5	14.5
problem 2					
# iter.	100	50	26	15	12
conv. rate	0.83	0.68	0.49	0.25	0.20
time (sec.)	199.2	101.1	52.9	30.6	24.6

3.3.2 Numerical methods

At numerical experiments, smoothing method of the MGCG method is one iteration of the Red-Black symmetric Gauss-Seidel (RB-SGS) smoothing. The RB-SGS smoothing has inherent high parallelism. Therefore the MGCG method with this highly parallel smoothing can be efficiently implemented on distributed memory machines.

We are interested in only the convergence rate of the MGCG method with fewer grid levels, thus this numerical experiment is performed on a serial machine, HP9000/720, and the program is written in C++ with highly optimized vector and matrix classes.

3.3.3 Convergence rate of the MGCG method

Table 1 is the result of these numerical experiments. The convergence condition is $\frac{\|r_m\|_2}{\|r_0\|_2} < 10^{-8}$, where r_m is the residual after m iterations.

From this observation, the average convergence rate of the MGCG method depends on the mesh size of the coarsest grid. In other words, the number of iterations until convergence diminishes a half as the number of grid levels increases. Note that the standard multigrid method is not robust on the problem 2.

Consequently, the greater number of grid levels the MGCG method uses, the better convergence rate it gets. The average convergence rate becomes square as the number of grid levels increases. The time until convergence of the MGCG method is also shortest when 7 grid levels are used on both problems.

3.4 Eigenvalue analysis

In order to study the efficiency of the multigrid preconditioner with fewer grid levels, we examine the eigenvalue distribution of a coefficient matrix after the multigrid preconditioning. The multigrid preconditioner uses the highly parallel RB-SGS smoothing. On the coarsest grid, only the RB-SGS smoothing is performed. The number of iterations of the CG method until convergence depends on the number of eigenvalues of eigenvectors included in the initial residual vector. Therefore we evaluate the number of iterations of the worst case by investigating eigenvalue distribution. The problem is problem 2 of Subsection 3.3, and the domain is discretized into the meshes of 16×16 by the finite element method. The condition number is 6.05e+3.

[15] describes how to calculate a matrix after the multigrid preconditioning.

The eigenvalue distributions after the multigrid preconditioning and the IC(1, 2) preconditioning are shown by Figures 5 and 6, respectively. The horizontal x axis is the order of the eigenvalues and the vertical y axis is the eigenvalues. This multigrid preconditioner uses only 2 grid levels instead of 4, and it does not exactly solve a linear equation in the coarse grid correction on the coarsest grid, but relaxes it by the RB-SGS smoothing.

The eigenvalue distribution of the multigrid preconditioned matrix is effective for the CG method as the following points:

1. Almost all eigenvalues are clustered around 1, and few small isolated eigenvalues exist between 0 and 1.

2. The smallest eigenvalue is larger than that of the ICCG method, and condition number is improved.

The first item causes no problem for the CG method since each eigenvector included in residual vector corresponding to these scattered eigenvalues is vanished in each CG iteration. All these characteristics are

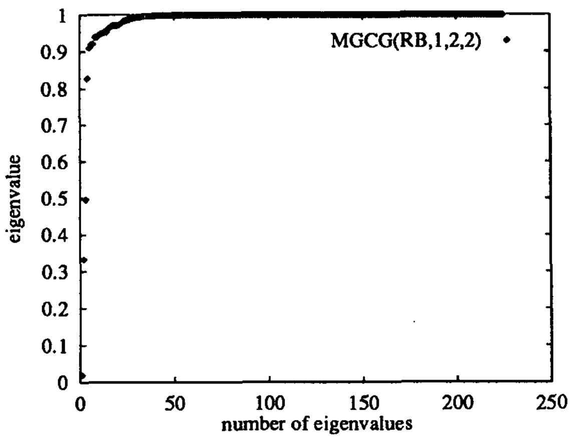

Figure 5: Eigenvalue distribution after the multigrid preconditioning

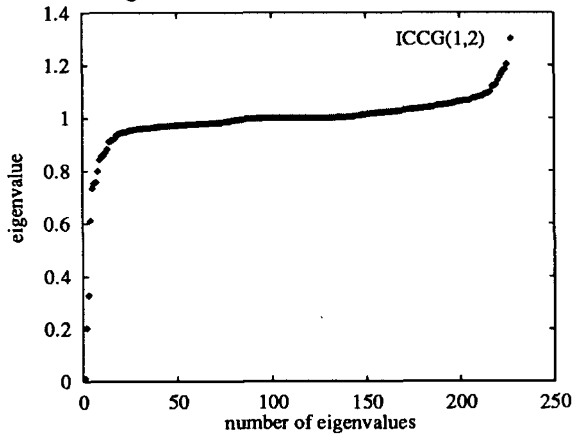

Figure 6: Eigenvalue distribution after the IC(1,2) preconditioning

desirable to accelerate the convergence of the CG method. Therefore the multigrid preconditioner is efficient to the CG method even if it uses a small number of grid levels, and the convergence rate of the MGCG method does not deteriorate suddenly.

4 Implementation on AP1000

This section evaluates the MGCG method on Fujitsu multicomputer AP1000[8]. AP1000 is a two-dimensional torus connected parallel computer with distributed memory. A node processor is 25MHz SPARC chip and FPU. Each node has a 128KB cache memory and a 16MB local memory. Each port of torus network has 25MB/s data transfer rate.

```
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    b[i][j] = dd[i][j][2] * a[i - 1][j] +
              dd[i][j][1] * a[i][j - 1] +
              dd[i][j][0] * a[i][j] +
              dd[i][j][3] * a[i][j + 1] +
              dd[i][j][4] * a[i + 1][j];
```

Figure 7: Fragment of pentadiagonal matrix-vector multiplication

4.1 Implementation techniques on distributed memory machines

This subsection describes implementation techniques for highly parallel efficiency on distributed memory machines.

4.1.1 Overlapping communication with computation

When the data is distributed by block (or tiling), the MGCG method needs nearest-neighbor communication in matrix-vector multiplication, the RB-SGS smoother, restriction and prolongation. Figure 7 is an example of a fragment of pentadiagonal matrix-vector multiplication. These computations can be separated into two parts: computation of inner points and computation of edge points (Figure 8), since the communication pattern is described only by constant distance vector[2, 9]. Computation of the inner points does not need data owned by other processors, thus it can work locally. Computation of the edge points needs external data, thus it is not completed without receiving them. If external data reach to this processor while the inner points are computed, the network latency is hidden. The nodal code of Figure 7 in two processors case is illustrated by Figure 8. For example, if `b[i][j]`, `a[i][j]` and `dd[i][j][0..4]` are all allocated to the same processor, a pentadiagonal matrix-vector multiplication needs the data `a[i - 1][j]`, `a[i][j - 1]`, `a[i][j + 1]` and `a[i + 1][j]`. In this case, the number of the inner points is $O(NM)$ and the number of the edge points is $O(\sqrt{N} + \sqrt{M})$, where the allocated region is $N \times M$ meshes. Thus the network latency is better hidden when N and M are larger. This technique is popular and efficient optimizing technique for data-parallel compiler (for example [5, 9]).

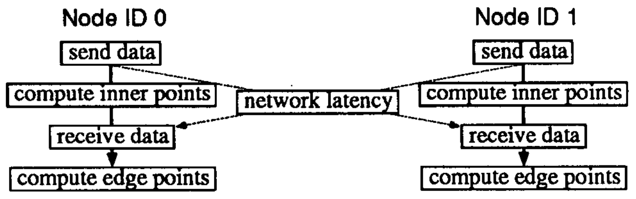

Figure 8: Nodal code for overlapping communication with computation

4.1.2 Efficient cache utilization

Cache memory increases as the number of processors of distributed memory machines increases. Thus clever utilization of cache memory is quite important. For example, the two possibilities of data allocation of pentadiagonal coefficient matrix are considered. One definition is

```
double dd1[N][M][5];
```

The other definition is

```
double dd2[5][N][M];
```

Pentadiagonal matrix-vector multiplication using the former matrix definition is coded by Figure 7. If the latter is used, pentadiagonal matrix-vector multiplication is programmed like the following code:

```
for (k = 0; k < 5; k++)
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[i][j] += dd[i][j][k] *
                a[i - mi[k]][j - mj[k]];
```

This program can stand further improvement of range of the indexes i and j, however it is a trivial improvement. The most serious difference between the two is the access order of the array a. The number of accesses to the whole array a is almost only one in the former, while that is five in the latter. Five accesses cause more cache miss. The cache miss reduces the cache effect that is discussed by Subsection 4.6. Difference of performance of these access orders is more notable as the number of processors increases.

4.2 Data distribution

Data distribution is particularly critical for distributed memory machines and it is crucial to performance. Matrix-vector multiplication, restriction, prolongation and smoothing methods need nearest-neighbor communication. Therefore block (or tiling) distribution is desirable. In the following discussion, we only consider in two-dimensional case. When the number of processors is M, possible distribution methods are the following (A) and (B):

(A) Vector is distributed logically to $M \times 1$. In HPF manner, distribution directive is

```
!HPF$ DISTRIBUTE (BLOCK, *)
```

(B) Vector is distributed logically to $\sqrt{M} \times \sqrt{M}$. In HPF manner,

```
!HPF$ DISTRIBUTE (BLOCK, BLOCK)
```

When problem size is N, the computational complexity is $O(N/M)$ in both methods since it is proportional to the number of grid points in each processor. On the other hand, since the number of communication data is proportional to the number of grid points included in subdomain's edge, that of the method (A) is $O(\sqrt{N})$ and that of the method (B) is $O(\sqrt{N}/\sqrt{M})$. The communication latency of the method (A) is smaller than that of the method (B). Thus there is a trade-off between N and M. When M is comparatively small with N, the method (A) is advantageous, and when M is large, the method (B) is advantageous.

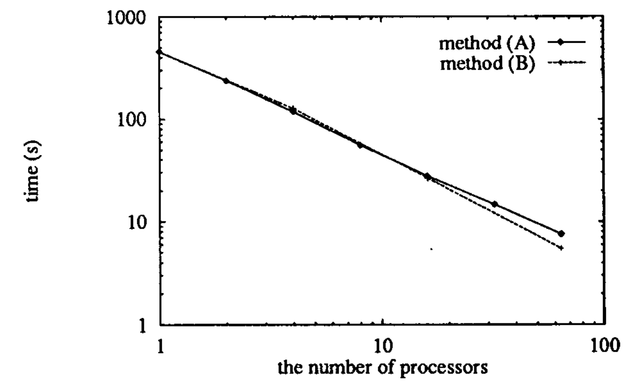

Figure 9: Comparison between (A) and (B)

Figure 9 compares the methods (A) and (B) on AP1000. In this numerical experiment, the problem is problem 1 of Subsection 3.3, and the MGCG method exploits only two grid levels. In 4 processors case, the method (A) is faster than the method (B). The method (A), however, begins to be saturated over 16 processors. Therefore the method (B) is advantageous on massively parallel machines. In the following discussion, data is distributed by the method (B).

4.3 Optimal number of grid levels

In Subsection 3.3, the average convergence rate of the MGCG method that uses fewer grid levels is considered. It is concluded that the greater number of grid levels the MGCG method uses, the better convergence rate it gets on Poisson's equation. This subsection considers optimal number of grid levels of the MGCG method on the multicomputer AP1000. As the number of grid levels increases, the convergence rate of the MGCG method becomes better. On the other hand, communication and synchronization overheads increase. Therefore there is the trade-off among the number of grid levels.

The problem is problem 2 of Subsection 3.3. Table 2 is the relationship between the number of grid levels and the time until convergence. Smoothing method of the MGCG method is two iterations of the RB-SGS smoother. The multigrid cycle is V-cycle. Convergence condition is $\frac{\|r_m\|_2}{\|r_0\|_2} < 10^{-6}$, where r_m is the residual after m iterations. 64 (8×8) processors are used in this numerical experiment.

Table 2: V-cycle MGCG method with various # of grid levels (256×256)

grids	iter	time (sec.)	MFlops	coarsest
2	135	10.25	129.0	128×128
3	64	5.70	126.4	64×64
4	32	3.05	122.5	32×32
5	17	1.71	117.4	16×16
6	10	<u>1.05</u>	113.0	8×8

In these tables, 'coarsest' column means the number of meshes on the coarsest grid. When this column is 8×8, one processor owns substantially only one datum on the coarsest grid.

From this result, the time until convergence is shorter as the number of grid levels increases, while MFlops decreases by increasing the communication overhead. Thus the MGCG method with 6 grid levels is fastest in 256 × 256 case. This result is the same on serial computer in Subsection 3.3. This is explained as follows. On coarser grid, since there are few data in each processor, the communication overhead is significant. On the other hand, since the number of grid points on coarser grid is one quarter of that on finer grid, there are few computational works on coarser grid. Also, data are locally manipulated. That is, working set of memory reference becomes small. Thus cache hit ratio is high, and hit ratio of line sending is also high. Line sending is a special optimization of communication on AP1000; it sends data directly from the cache memory if data hits in the cache memory, and this mechanism enables small message handling overhead. Therefore the time per 1 iteration does not increase very much when one more coarser grid is added. Consequently, the improvement of the convergence rate of the MGCG method surpasses the deterioration of MFlops on AP1000.

4.4 Optimal number of iterations of the smoothing method

This subsection considers optimal number of iterations of the smoothing method. Problem and the smoothing method of the MGCG method are the same of the previous subsection. Table 3 is the result.

Table 3: V-cycle MGCG method with various # of smoothing iterations (256×256)

grids	5			6		
#sm.	it.	time	MFlops	it.	time	MFlops
1	22	1.62	111.6	13	<u>1.00</u>	107.6
2	17	1.71	117.4	10	1.05	113.0
3	14	1.79	120.8	9	1.20	116.2

From this result, when the number of smoothing iterations increases, MFlops also increases. The convergence rate is improved of course. However, the improvement of MFlops and the convergence rate does not surpass the increase of the time per one iteration. Besides, the convergence rate is not improved by the greater number of smoothing iterations than by the greater number of grid levels. Therefore the time until convergence is short when the greater number of grid levels and one smoothing iteration are used.

4.5 Optimal cycle of the multigrid preconditioner

The popular multigrid cycles are V- and W-cycles. The W-cycle multigrid method has many coarse grid corrections, and it can solve more complex problem efficiently than the V-cycle multigrid method. This subsection investigates the MGCG method with the W-cycle multigrid preconditioner.

Table 4 is the result. Comparing with the V-cycle multigrid preconditioner, MFlops increases incredibly. That is because the first recursive call of the multigrid

Table 4: W-cycle MGCG method (256×256)

grids	5			6		
#sm.	it.	time	MFlops	it.	time	MFlops
1	9	<u>1.26</u>	160.6	7	1.38	120.7
2	8	1.60	167.9	6	1.71	124.9
3	7	1.82	172.0	5	1.87	127.3

function needs an zero initial vector, and the second call does not need, cf. Figure 2. Thus zero initializing is dispensable, and this contributes the improvement of MFlops. The W-cycle MGCG method is faster than V-cycle when the number of grid levels is 4 and 5. However, the W-cycle MGCG method with 6 grid levels is slower even than the W-cycle MGCG method with 5 grid levels. That is because W-cycle uses the coarsest grid most frequently in all grid levels, while V-cycle uses only one time. Therefore the communication overhead is crucial to performance.

Consequently, the MGCG method whose number of the coarsest grid points is equal to the number of processors, whose number of smoothing iterations is 1 and whose cycle of the multigrid preconditioner is V-cycle, is the fastest method on AP1000. It is a quite satisfactory result and it implies that communication and synchronization overheads are able to be hidden by clever implementation, such as overlapping communication with computation.

4.6 Performance evaluation

In the previous subsections, we consider the efficient MGCG method on the multicomputer AP1000. This subsection evaluates the MGCG method in terms of speedup on fixed problem size and fixed ratio of the problem size and the number of processors. In the fixed problem size case, speedup is calculated by both time and *million floating-point operations per second* (MFlops). When p processors are used, time speedup S_p^{time} and MFlops speedup S_p^{MFlops} are calculated by

$$S_p^{\mathrm{time}} = \frac{T_1}{T_p} \text{ and } S_p^{\mathrm{MFlops}} = \frac{M_p}{M_1},$$

where T_i is the calculation time and M_i is MFlops with i processors. In the fixed problem size case, scalar computations are computed on every processor in our program, the number of floating point operations slightly increases proportionally to the number of processors. Thus $S_p^{\mathrm{MFlops}} \geq S_p^{\mathrm{time}}$.

Figure 10 is the result in the case of the fixed problem size. From this figure, superlinear speedup

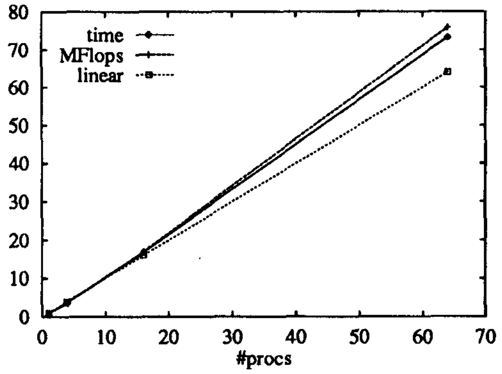

Figure 10: Speedup (fixed problem size)

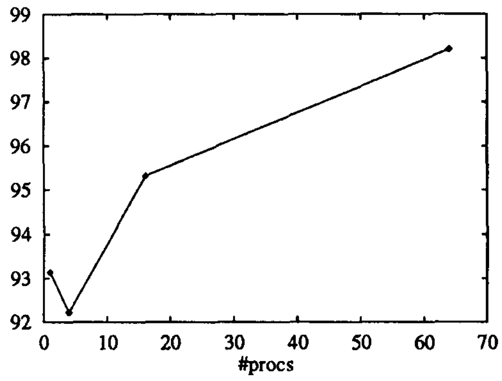

Figure 11: Cache hit ratio

is observed. According to Amdahl's law, superlinear speedup never happens. However, as the number of processors increases, data size of each processor is smaller since the total data size is fixed. Besides, in the MGCG method, data size on coarse grids is small. Thus cache hit ratio and hit ratio of line sending become high. Cache hit ratio is shown by Figure 11. From this observation, cache hit ratio increases as the number of processors increases except 4 processors case. That is because 4 processors case needs interprocessor communication, and received data always cause cache miss though storage of cache memory is four times larger than one processor case.

Next we consider the case of the fixed ratio of the problem size and the number of processors. In this case, since the problem size increases proportionally to the number of processors, time speedup cannot be calculated. Thus speedup in MFlops is only calculated. The previous case is effected by the cache memory, since cache size increases proportionally to the number of processors. However, this case are not influenced by the cache effect due to fixing ratio of the problem size and the number of processors. It is ideal that the ratio of the number of floating point instructions and the number of processors is fixed. However, we consider the number of floating point instructions is approximately proportional to the problem size, since the convergence rate of the MGCG method depends on the mesh size of the coarsest grid. Therefore the evaluation by the fixed ratio of the problem size and the number of processors is plausible.

Table 5: Performance of the MGCG method

#procs.	1	4	16	36	64
size	32^2	64^2	128^2	192^2	256^2
iter.	5	5	6	6	7
time (s.)	0.266	0.346	0.466	0.467	0.543
MFlops	2.30	7.34	26.6	60.0	107.1
speedup	-	3.19	11.6	26.1	46.6
effic.	-	0.798	0.722	0.724	0.728

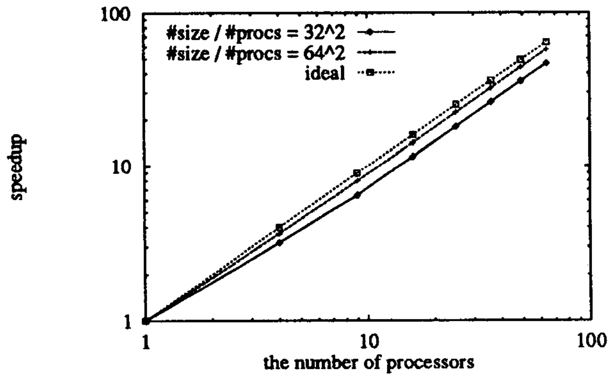

Figure 12: Speedup (fixed ratio of problem size and # processors).

Table 5 is the result of numerical examples. 'effic.' row means the parallel efficiency. It is calculated by $\frac{S_p^{\text{MFlops}}}{p}$, where p is the number of processors. Figure 12 is a speedup graph. This numerical experiment is performed on Poisson's equation with constant diffusion coefficient. Source term is a constant value. We use MGCG(RB, 1, 1, 5) method for 1 processor case and MGCG(RB, 1, 1, 6) method for 4 or more processors case. That is because 1 processor case exploits only 5 grid levels.

Though the cache effect is not expected in this case, almost proportional speedup is observed. Since 4 processors case needs communication among nearest-neighbor processors, the efficiency is inferior to 1 processor case. While 4 processors case needs communication in only 2 or 3 directions, 9 or more processors case needs communication in 4 or 8 directions. Therefore 9 or more processors case is still inferior to 4 processors case. When the number of processors exceeds 9, the parallel efficiency is slightly better. That is because the number of floating point operations increases since the number of iterations until convergence slightly increases.

4.7 Comparison with the SCG method

Finally the SCG method is compared with the MGCG method. The SCG method is a CG method with diagonal scaling preconditioner. This method is 100% parallelizable since diagonal scaling does not have to require any communication. Thus completely proportional speedup is expected. It has been reported that total convergence time on vector machines is shorter than that of the ICCG method, though the SCG method needs much more iterations[6]. That is because the ICCG method requires the solution of triangular systems, and this solution is too expensive on vector machines.

Table 6: Comparison with the SCG method

size	iter.	time (s.)	MFlops	SCG/MGCG
256^2	972	14.6	95.28	14.6 times
512^2	1954	155.7	72.11	36.4 times

Table 6 is the result comparing with the MGCG method. The problem is problem 2 of Subsection 3.3. The SCG method has high parallelism, and it gains high parallel efficiency. However the number of iterations until convergence is 75 times to 150 times larger than that of the MGCG method. Since the MGCG method also has high parallelism, the time until convergence of the SCG method is 15 times to 36 times longer than that of the MGCG method.

Although the SCG method is perfectly parallelizable, a poor convergence rate is critical. Moreover, the cache effect described in Subsection 4.6 is not exploited when the problem size is larger, in contrast to the MGCG method.

5 Conclusion

We have studied the parallelization of the MGCG method and efficient implementation on distributed memory machines. Since the MGCG method is a combination of the CG method and the multigrid method, both the CG method and the multigrid method should be parallelized. There are many problems in parallelizing the multigrid method. However, the MGCG method is efficiently parallelized on distributed memory machines, since the MGCG method with highly parallel point relaxation is robust. The rest is an implementation matter such that comparatively coarse grid with the number of processors causes low performance. By implementing on the multicomputer AP1000, it is shown that the MGCG method is an efficient method on distributed memory machines, if the multigrid preconditioner uses sufficient number of grid levels such that it gains good convergence rate and it does not cause low performance. Moreover, by careful implementation, the parallel MGCG method shows almost proportional speedup to the number of processors. It is more than 10 times faster than the SCG method on AP1000.

Since the MGCG method has high parallelism and fast convergence, this method is a very promising method as the solution of a large-scale sparse, symmetric and positive definite matrix on not only serial machines but distributed memory machines.

Acknowledgments

We would like to thank the members of Fujitsu Parallel Computing Research Facilities for providing and supporting AP1000 multicomputer. We appreciate valuable discussions of all members of Oyanagi laboratory at the University of Tokyo. This work is partly supported by the Grand-in-Aid of Ministry of Education, Science and Culture of the Japanese Government (Nos. 04235101 and 06680305).

References

[1] Alcouffe, R. E., A. Brandt, J. E. Dendy Jr., and J. W. Painter, "The multi-grid method for the diffusion equation with strongly discontinuous coefficient," *SIAM J. Sci. Stat. Comput.*, vol. 2, pp. 430–454, 1981.

[2] Amarasinghe, S. P. and M. S. Lam, "Communication Optimization and Code Generation for Distributed Memory Machines," in *Proceedings of ACM Conference on Programming Language Design and Implememtation*, pp. 126–138, June 1993.

[3] Dendy Jr., J. E., "Black box multigrid," *J. Comp. Phys.*, vol. 48, pp. 366–386, 1982.

[4] Dendy Jr., J. E., M. P. IDA, and J. M. Rutledge, "A Semicoarsening Multigrid Algorithm for SIMD Machines," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 6, pp. 1460–1469, November 1992.

[5] Hatcher, P. and M. Quinn, *Data-parallel programming on MIMD computers.* MIT Press, 1991.

[6] Hayami, K. and N. Harada, "The Scaled Conjugate Gradient Method and Vector Processors," in *Proceedings of the First International Conference on Supercomputing Systems*, (St. Petersburg, Florida), pp. 213–221, IEEE Computer Society, 1985.

[7] Hempel, R. and M. Lemke, "Parallel Black Box Multigrid," in *Proceedings of the fourth Copper Mountain Conference on Multigrid Method* (J. Mandel et al, ed.), pp. 255–272, April 1989.

[8] Ishihata, H., T. Horie, S. Inano, T. Shimizu, and S. Kato, "An Architecture of Highly Parallel Computer AP1000," in *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, pp. 13–16, May 1991.

[9] Koelbel, C. and P. Mehrotra, "Compiling Global Name-Space Parallel Loops for Distributed Execution," *IEEE trans. of Parallel and Distributed Systems*, vol. 2, no. 4, pp. 440–451, October 1991.

[10] Mulder, W. A., "A New Multigrid Aproach to Convection Problems," *J. Comp. Phys.*, vol. 83, pp. 303–323, 1989.

[11] Naik, N. H. and J. V. Rosendale, "The Improved Robustness of Multigrid Elliptic Solvers Based on Multiple Semicoarsened Grids," *SIAM J. Numer. Anal.*, vol. 30, no. 1, pp. 215–229, February 1993.

[12] Overman, A. and J. V. Rosendale, "Mapping Robust Parallel Multigrid Algorithms to Scalable Memory Architectures," in *Proceedings of Sixth Copper Mountain Conference on Multigrid Methods*, pp. 635–647, NASA Conference Publication 3224, April 1993.

[13] Sbosny, H., "Domain Decomposition Method and Parallel Multigrid Algorithms," in *Multigrid Methods: Special Topics and Applications II* (W. Hackbusch and U. Trottenberg, eds.), no. 189 in GMD-Studien, pp. 297–308, May 1991.

[14] Stüben, K. and U. Trottenberg, "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications," in *Multigrid Methods, Proceedings of the Conference Held at Köln-Porz* (W. Hackbusch and U. Trottenberg, eds.), vol. 960 of *Lecture Notes in Mathematics*, pp. 1–176, Springer-Verlag, 1982.

[15] Tatebe, O., "The Multigrid Preconditioned Conjugate Gradient Method," in *Proceedings of Sixth Copper Mountain Conference on Multigrid Methods*, pp. 621–634, NASA Conference Publication 3224, April 1993.