Malaysian Journal of Computer Science, Vol. 13 No. 2, December 2000, pp. 1-6

A PARALLEL HALFSWEEP MULTIGRID ALGORITHM ON THE SHARED MEMORY MULTIPROCESSORS

Mohamed Othman
Department of Communication Technology and Networks
Universiti Putra Malaysia
43400 UPM Serdang, Selangor D.E.

J. Sulaiman
School of Science and Technology
Universiti Malaysia Sabah
Kota Kinabalu, Sabah

A. R. Abdullah
Department of Industrial Computing
Universiti Kebangsaan Malaysia
43600 UKM Bangi, Selangor D. E.
Malaysia

ABSTRACT

The halfsweep multigrid algorithm, introduced by Othman et al in 1998 for solving a linear system, is known as a fast multigrid poisson solver. In this paper, the implementation of the parallel halfsweep multigrid algorithm with several parallel strategies is discussed. The experiments were carried out on the shared memory multiprocessors computer system, Sequent S27, and the results of the test problem are included.

Keywords: ***Parallel halfsweep multigrid algorithm; Parallel strategy; Performance evaluation***

1.0 INTRODUCTION

Multigrid method has been known for many years. It is fast and one of the most efficient iterative methods for solving a wide variety of scientific computing and engineering problems. Despite advances in computer hardware, many applications require still greater performance than that offered by traditional computers. Given the success of the sequential multigrid algorithm, the V(1, 1)-cycle halfsweep multigrid algorithm (introduced by Othman *et al* in 1998), it is natural to consider the parallel version of the algorithm, especially, on the shared memory multiprocessors platform.

In the case of the fullsweep approach, several successful parallel multigrid algorithms have been implemented on various parallel computer platforms [1, 2, 3, 4, 6]. For instance, Chan *et al* [1] implemented the parallel multigrid algorithm on the Hypercube Multiprocessor computer system.

2.0 FULLSWEEP MULTIGRID METHOD

The fullsweep multigrid method has been used by many researchers. It employs all the points (or tasks) at any level of the hierarchical grid (i.e. Ω^h, Ω^{2h}, ..., Ω^{Nh}) for their computations. The method uses the three points stencil, as a grid smoother coincide with the Gauβ-Seidel chess board strategy for their pre- and post- smoothing stages. Since all the tasks at each level of the hierarchical grid are involved in the computations, the full weighting restriction operator is used to transfer all the calculated residuals from fine Ω^h to coarser grid Ω^{2h} defined as,

$$R_h^{2h} = \tfrac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}.$$

On the other hand, the bilinear prolongation operator P_{2h}^{h} is used to transfer the error corrections from coarse Ω^{2h} to finer grid Ω^h given by,

$$v_{2i}^{h} = v_i^{2h}, \qquad 0 \le i \le N_c$$
$$v_{2i+1}^{h} = \tfrac{1}{2}\left(v_i^{2h} + v_{i+1}^{2h}\right), \qquad 0 \le i \le N_c - 1$$

where N_c is the size of the coarser grid. Briefly, the $V(\eta_1,\eta_2)$-cycle fullsweep multigrid algorithm is described in C-like language as shown in Appendix 1.

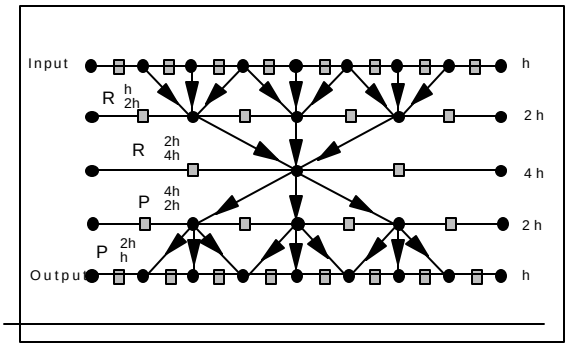

Fig. 1: The graphical structure of the V(η_1, η_2)-cycle halfsweep multigrid method

3.0 HALFSWEEP MULTIGRID METHOD

According to Othman and Abdullah [5], all tasks at any levels of the hierarchical grid, (i.e. Ω^{h}, Ω^{2h}, ..., Ω^{Nh}) are labeled in chess board labeling, as shown in Fig. 1. A group of black (• tasks) will be computed using the three points stencil of width 2h until the convergence criteria are met, then the rest of red (tasks) will be executed at once using the three points stencil of width h, otherwise, the computation cycle is repeated. It shows that a group of black tasks can be implemented by involving only black tasks and the same happens for a group of red tasks. Therefore, the implementation of these two groups of tasks can be carried out independently and the execution time can be saved nearly by half if the computation over the hierarchical grid is only carried out on either group of tasks.

As only a group of black tasks are involved in the computation, the following restriction operator R_{h}^{2h} is required for transferring the calculated residuals from fine Ω^{h} to coarser grid Ω^{2h} given by,

$$R_{h}^{2h} = \tfrac{1}{4}\begin{bmatrix} 1 & 0 & 2 & 0 & 1 \end{bmatrix}.$$

All the error corrections of black tasks are transferred from coarse Ω^{2h} back to finer grid Ω^{h} defined by the following bilinear prolongation operator P_{2h}^{h},

$$v_{2i}^{h} = v_{i}^{2h}, \qquad \forall\, i = 0, 2, 4, \ldots, N_c$$

$$v_{2i+2}^{h} = \tfrac{1}{2}\left(v_{i+2}^{2h} + v_{i}^{2h}\right), \quad \forall\, i = 0, 2, 4, \ldots, N_c - 2$$

The chess board Gauβ-Seidel relaxation scheme is used as grid smoother for their pre- and post- smoothing stages. It is used to smooth the calculated residuals and error corrections at the coarse grids. Appendix 1 describes the nested V(η_1,η_2)-cycle halfsweep multigrid algorithm.

4.0 STRATEGIES AND THEIR PARALLEL IMPLEMENTATIONS

Since all the black tasks at any level of the hierarchical grid are identical, the data partitioning approach is suitable in implementation of the methods. All the identical tasks can be executed in parallel, and again, the static scheduling is also employed.

Three main procedures involved in the implementation are described in the following sections.

4.1 Parallel Grid Smoother

The Gauβ-Seidel relaxation scheme is used as a grid smoother due to the fact that the new updated values are used to calculate the next value, as it becomes available. It is very important that the residuals are well smoothed before they can be transferred to the coarser or finer grids.

Since data dependence among the tasks occurred at any level of the hierarchical grid, the chess board strategy is employed in the smoother and each task is allocated to a processor at a time. Thus, every processor independently computes its own tasks in parallel. The C-like language codes below show the parallel grid smoother with the chess board strategy.

```
Par_grid_smoother_procedure()
{nprocs=m_get_numprocs();
  id=m_get_myid(); inc=2*nprocs;

 for (color=0; color<=1;color++) {
   if (color==0) s=2+4*id;
   else       s=4+4*id;
   for (i=s;i<Nc;i=i+inc) u[i]=0.5*(u[i-2]+u[i+2]-2h2*f[i]);
   m_sync();
}}
```

Once the convergence criteria is met, no data dependency occurs at the finest grid, then all the red tasks are smoothed at once in parallel by employing the natural strategy. The Par_grid_direct_procedure() shows the smoother of the parallel direct relaxation scheme.

```
Par_grid_direct_procedure()
{ nprocs=m_get _numprocs(); id=m_get_myid();
   inc=2*nprocs;

   for (i=1+id;i<Nc;i=i+inc)
       u[i]=0.5*(u[i-1]+u[i-1]- h2f[i]);
   m_sync();
}
```

4.2 Parallel Restriction Operator

In the restriction procedure, there are two main computations which depend on each other. They are the computations of residual and full weighting restriction. These computations must be executed one after another, while the synchronization call at the end of each computation ensures that the updated values are used in the second computation. Due to the fact that no data dependency occurs in each computation, the individual computation can be executed in parallel by employing the natural strategy. Each task from each computation is assigned to one processor at a time, and then every processor independently computes its own tasks. These computations are shown in the following C-codes.

```
Par_restriction_procedure()
{  nprocs=m_get_numprocs(); id=m_get_myid();
   inc=2*nprocs; half= 0.5*Nc;

   for (i=2+2*id;i<Nc;i=i+inc)
     w[i]=0.5h-2*(2*u[i]-u[i-2]-u[i+2])-f[i];
   m_sync();
   for (k=2+2*id;k<half; k=k+inc) {
     i=2*k;
     y[k]=0.25*(w[i-2]+w[i+2]+2*w[i]);
   }
   m_sync();
}
```

4.3 Parallel Prolongation Operator

There are two main computations involved in the prolongation procedure, they are the computation of prolongation and bilinear operation. These two computations must be executed one after another as the second computation depends on the results of the first computation. The synchronization call at the end of each computation ensures that the updated values are available for the following computation. In the individual computation, no data dependency occurs among the tasks, thus, they can be executed in parallel by employing the natural strategy. The procedure of these computations is shown in the following C-like language codes below:

```
Par_prolongation_procedure()
{  nprocs=m_get_numprocs(); id=m_get_myid();
   inc=2*nprocs; half= 0.5*N_c;

   for (i=2+2*id;i<half; i=i+2*inc) w[2*i]=u[i];
   m_sync();

   for (i=2+2*id;i<N_c;i=i+inc)  w[i]=0.5*(w[i-2]+w[i+2]);
   m_sync();
}
```

4.4 Parallel Halfsweep Multigrid Algorithm

The parallel V(η_1, η_2)-cycle halfsweep multigrid algorithm is described in C-like language as stated in Appendix 2.

5.0 PERFORMANCE EVALUATION

In order to confirm that the parallel halfsweep multigrid algorithm is superior to the parallel fullsweep multigrid algorithm, the following experiments are carried out on the shared memory multiprocessor computer system, Sequent S27. All the methods were applied to the following test problem ($u_{xx} = -x$) in a unit cartesian region, subject to the Dirichlet condition. To avoid time taken for system, user and other I/O overheads, the algorithms were executed when no other users were using the computer. Throughout the experiments, all the algorithms were carried out on different sizes of finest grids 2^{13}, 2^{14}, 2^{15} and 2^{16} with V(η_1,η_2)-cycle. The algorithms will stop when all tasks at the finest grid, which undergo the computation, are less than $\varepsilon=10^{-10}$.

The experimental results are reported in the Table 1. The graphs for execution time, speedup and efficiency versus number of processors were plotted and shown in Figs. 2, 3 and 4, respectively. The temporal performance is usually used to compare the performance of different algorithms for solving the same problem and it is defined as,

$$P_p = \frac{1}{T_p}$$

where the unit is work done per second, and p is the number of processors. The algorithm with the highest performance executes in the least time and, therefore, is the better algorithm. Fig. 5 shows the graph of the temporal performance versus number of processors for $n=2^{16}$.

Table 1: The execution time, speedup and efficiency of the parallel multigrid algorithms with full- and half- sweep approaches

n	No. of procs	Full			Half		
		Time	Speedup	Efficiency	Time	Speedup	Efficiency
2^{13}	1	4.45	1.00	1.00	2.73	1.00	1.00
	2	2.96	1.50	0.75	2.05	1.33	0.66
	3	2.41	1.84	0.61	1.67	1.63	0.54
	4	2.22	2.00	0.50	1.58	1.72	0.43
	5	1.99	2.23	0.44	1.51	1.80	0.36
2^{14}	1	8.27	1.00	1.00	5.51	1.00	1.00
	2	5.30	1.56	0.78	3.59	1.53	0.76
	3	3.89	2.12	0.70	2.81	1.95	0.63
	4	3.42	2.41	0.60	2.41	2.28	0.57
	5	3.11	2.65	0.53	2.27	2.42	0.48
2^{15}	1	16.00	1.00	1.00	10.47	1.00	1.00
	2	9.90	1.61	0.80	6.56	1.59	0.79
	3	7.21	2.21	0.73	4.78	2.18	0.72
	4	6.53	2.45	0.61	4.33	2.41	0.60
	5	5.39	2.96	0.59	3.72	2.80	0.56
2^{16}	1	31.70	1.00	1.00	20.44	1.00	1.00
	2	18.41	1.72	0.86	12.58	1.62	0.81
	3	13.15	2.41	0.83	9.43	2.16	0.72
	4	10.93	2.90	0.71	7.69	2.65	0.66
	5	9.73	3.25	0.65	6.47	3.15	0.63

6.0 CONCLUSION

Based on Table 1 and Fig. 1, the results show that the parallel halfsweep multigrid algorithm with the chess board Gauβ-Seidel grid smoother is superior to the parallel fullsweep multigrid algorithm for any number of processors, as n gets larger. This is due to the lower total computational operations in the algorithm as approximately half of the total tasks in each level are involved in the computation. In view of this, we found that the speedup and efficiency of the parallel halfsweep multigrid algorithm are not as good as that for the other algorithm. It can be improved by increasing the grid size *n*, (refer to Figs. 2 and 3). Furthermore, the superiority of the parallel halfsweep algorithm is also indicated by the highest value of the temporal performance (see Fig. 4).

In conclusion, the parallel halfsweep multigrid algorithm with the chess board strategy is the more effective algorithm when compared to the parallel fullsweep multigrid algorithm.

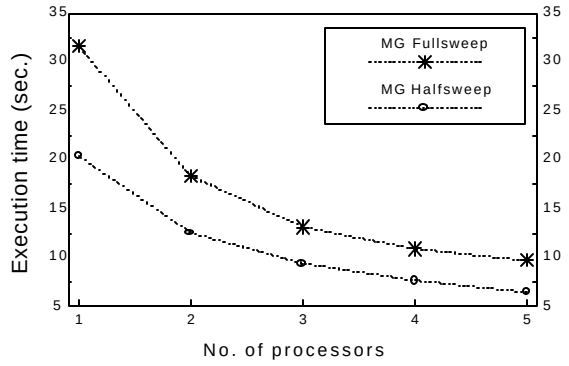

Fig. 2: Execution time versus no. of processors for $n=2^{16}$

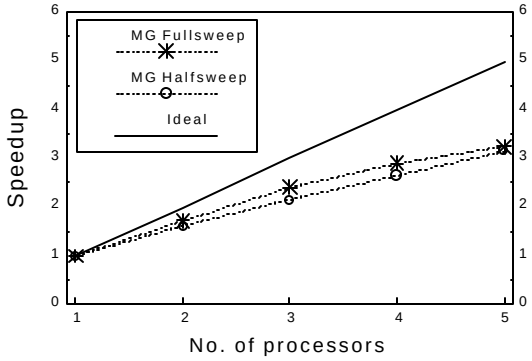

Fig. 3: Speedup versus no. of processors for $n=2^{16}$

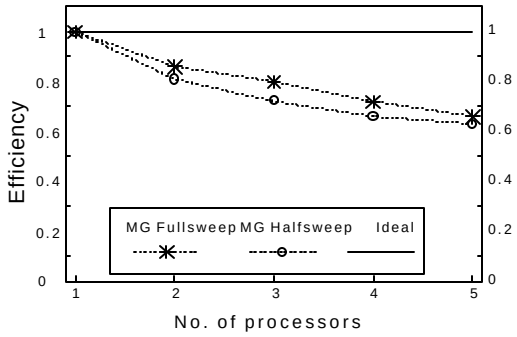

Fig. 4: Efficiency versus no. of processors for $n=2^{16}$

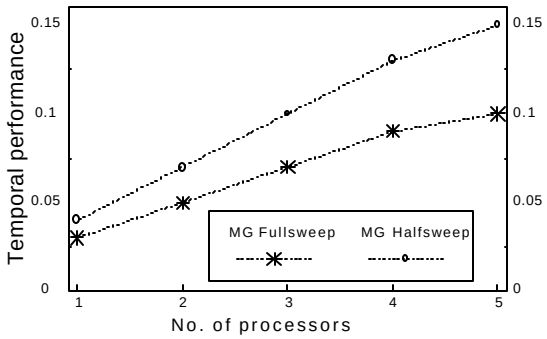

Fig. 5: Temporal performance versus no. of processors for $n=2^{16}$

REFERENCES

[1] T. F. Chan and Y. Saad. "Multigrid Algorithms on the Hypercube Multiprocessor". *IEEE Transaction on Computer*, Vol. C-35, No. 11, 1986, pp. 969-977.

[2] S. N. Gupta, M. Zubair and C. E. Grösch. "A Multigrid Algorithm for Parallel Computer: CPMG". *Journal of Scientific Computing*, Vol. 7, 1992, pp. 263-279.

[3] O. A. McBryan et al. "Multigrid Methods on Parallel Computers - A Survey on Recent Developments", *Impact of Computing in Science and Engineering*, Vol. 3, 1991, pp. 1-75.

[4] L. R. Matheson and R. E. Tarjan. "Parallelism in Multigrid Methods: How Much is Too Much?". *International Journal of Parallel Programming*, Vol. 24, No. 5, 1996, pp. 387-432.

[5] M. Othman and A. R. Abdullah. "The Halfsweeps Multigrid Method as a Fast Multigrid Poisson Solver". *International Journal of Computers and Mathematics*, Vol. 69, 1998, pp. 319-329.

[6] K. Sölchenbach, C. A. Thöle and U. Tröttenberg. "Parallel Multigrid Methods: Implementation of SUPRENUM-like Architectures and Applications". *INRIA Rapports de Recherche,* N°. 746, 1987.

BIOGRAPHY

Mohamed Othman obtained his Ph.D. from Universiti Kebangsaan Malaysia in 1999. Currently, he is a lecturer at the Department of Communication Technology and Networks, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. His research interest includes parallel computing, high speed network, cluster computing, artificial intelligence, expert system design, scientific computing and programming in logic (parallel). He has published over twenty technical papers related to his fields of research.

J. Sulaiman completed his MSc degree from Universiti Kebangsaan Malaysia in 1998. Currently, he is a lecturer at the School of Science and Technology, Universiti Malaysia Sabah. His research interest includes mathematical modeling, scientific computing and parallel computing.

A. R. Abdullah is a Professor in Industrial Computing at the Department of Computer Industry, Faculty of Computer Science and Information Technology, Universiti Kebangsaan Malaysia. His research interest includes parallel computing, high speed network computing and scientific computing. He has published a few books related to his fields of research.

Appendix 1: The nested V(η_1, η_2)-cycle fullsweep multigrid algorithm

```
SMGV(A^h, v^h, f^h) /* compute all the points until converge */
{
  if coarset grid, solve A^h e^h = r^h directly
  else {
    smooth η1 times on Gauβ-Seidel(A^h, v^h, f^h) using the three points stencil of width h
    compute residuals, r^h ← f^h - A^h v^h
    set e^2h ← 0, and restrict r^2h ← R (2h, h) r^h
    get e^2h ← SMGV(A^2h, v^2h, f^2h)
    compute prolongation and error (corr.), v^h ← v^h - P (h, 2h) e^2h
    smooth η2 times on Gauβ-Seidel(A^h, v^h, f^h) using the three points stencil of width h
  }
}
Algorithm Seq_halfsweep_mg()
{flag=0;
  while (flag != 1) do {
              flag=1;
              SMGV(A^h, v^h, f^h);
              if |v^(k+1) - v^(k)| > ε for all points, set flag=0
              iterate++;   swap all tasks, v^(k+1) → v^(k)
  }
  return v^h as an approximate solution
}
```

Appendix 2: The parallel V(η_1, η_2)-cycle halfsweep multigrid algorithm

```
DIRECT(A^h,v^h,f^h) /*compute a group f red tasks in parallel */
{
compute Par_grid_direct_procedure(A^h,v^h,f^h)
}
PMGV(A^h,v^h,f^h) /* compute a group of black tasks in parallel until converge */
{
  smooth η1 times on Par_grid_smoother_procedure(A^h,v^h,f^h)
  compute Par_restriction_procedure(r^h,e^2h,r^2h)
     smooth η1 times on Par_grid_smoother_procedure(A^2h,v^2h,f^2h)
     compute Par_restriction_procedure(r^2h,e^4h,r^4h)
                ⋮
                if coarset grid, solve A^Nh e^Nh = r^Nh
                ⋮
     compute Par_prolongation_procedure(e^4h,e^2h)
     smooth η2 times on Par_grid_smoother_procedure(A^2h,e^2h,r^2h)
  compute Par_prolongation_procedure(v^2h,v^h)
  smooth η2 times on Par_grid_smoother_procedure(A^h,v^h,f^h)
}
Algorithm Par_halfsweep_mg()
{Initialize();
  Set l_flag=1; g_flag=0; id=m_get_myid(); nprocs=m_get_numprocs(); bit=id^2;  stop=nprocs^2-1;
  /* compute the following while block in parallel */
  while (g_flag != stop) do {
                PMGV(A^h,v^h,f^h)
                if |v^(k+1) - v^(k)| > ε on the black tasks, set l_flag=0
                if (l_flag == 1) {
                  m_lock();   g_flag=g_flag + 1;        m_unlock();
                }
                <synchronize>
                iterate++;  l_flag=1; swap all black tasks, v^(k+1) → v^(k)
                <synchronize>
}
compute the DIRECT(A^h,v^h,f^h) procedure in paral lel
<synchronize>
m_kill_proc();
}
```