ISSN: 0128-7680
Pertanika J. Sci. & Technol. 8(2):161-174 (2000)

An Efficient Parallel Quarter-sweep Point Iterative Algorithm for Solving Poisson Equation on SMP Parallel Computer

Othman M.[a] and Abdullah A. R.[b]

[a]Department of Communication Technology and Network
Faculty of Computer Science and Information Technology
University Putra Malaysia
43400 UPM Serdang, Selangor Darul Ehsan, Malaysia
E-mail: mothman @fsktm.upm.edu.my
[b]Department of Industrial Computing, University Kebangsaan Malaysia
43600 UKM Bangi Selangor Darul Ehsan, Malaysia

Received: 13 October 1998

ABSTRAK

Satu algoritma lelaran titik terbaru yang menggunakan pendekatan suku-sapuan telah menunjukkan masa pelaksanaan yang lebih cepat jika dibandingkan dengan algoritma lelaran titik penuh- dan separuh- sapuan untuk menyelesaikan persamaan Poisson dua dimensi (Othman *et al.* (1998)). Walau bagaimanapun, dua algoritma terakhir sesuai diimplementasikan secara selari (Evans (1984) dan Ali *et al.* (1997)). Dalam makalah ini, pengimplementasian algoritma selari yang terbaru dengan menggunakan strategi papan catur pada komputer selari multipemproses simetri akan diterangkan. Keputusan eksperimen daripada satu masalah kajian dibandingkan dengan keputusan dua algoritma selari terakhir.

ABSTRACT

A new point iterative algorithm which uses the quarter-sweep approach was shown to be much faster than the full-and half- sweep point iterative algorithms for solving two dimensional Poison equation (Othman *et al.* 1998)). However, the last two algorithms were found to be suitable for parallel implementation (Evans 1984) and Ali *et al.* (1997)). In this paper, the parallel implementation of the new algorithm with the chessboard (CB) strategy on Symmetry Multi Processors (SMP) parallel computer was presented. The experimental results of a test problem were compared with the later two parallel algorithms.

Keywords : Poisson equation, Parallel algorithms, Chessboard strategy, Full-, half- and quarter-sweep approaches, Performance evaluation

INTRODUCTION

The parallel point iterative algorithm which incorporates the full-sweep approach for solving a large and sparse linear system has been implemented successfully by Barlow and Evans (1982), and Evans(1984) while the half-sweep approach was introduced by Abdullah (1991) for the derivation of the four points *EDG*

method. Since the *EDG* method is explicit, it is suitable to be implemented in parallel on any parallel computer. Yousif and Evans (1995) implemented the parallel four, six and nine points *EDG* methods for solving the two dimensional Poisson equation, while Ali and Abdullah (1997) implemented the parallel point iterative algorithms which use the full- and half-sweep approaches for solving the two dimensional diffusion-convection equation. All the parallel point and block iterative algorithms were implemented on MIMD Sequent B8000 computer system at Parallel Algorithm Research Center (PARC), Loughborouh University of Technology, United Kingdom. In the case of point iterative algorithm, the results obtained shown that the parallel point iterative algorithm which uses the half-sweep approach is relatively faster than the parallel full-sweep point iterative algorithm. This is due to the lower total computational operations in the algorithm since only half of the total points are involved in the iterations.

In a more recent development, Othman *et al.* (1998) introduced a new point iterative algorithm which uses the quater-sweep approach for solving the two dimensional Poisson equation on MIMD computer system, the Sequent S27. The experimental and analytical results obtained have shown that the algorithm is superior than the point iterative algorithms which use the full- and half-sweep approaches. As we know, the iterative algorithm requires a tremendous amount of computer time for solving a large and sparse linear system With the advent of new emerging parallel computers, the parallel implementation of the new algorithm when incorporated with the CB strategy will improve the performance of the algorithm.

THE POINT ITERATIVE ALGORITHMS

The solution of two dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad (x,y) \in \Omega^h, \tag{1}$$

in a unit square Ω^h with Dirichlet boundary condition using the finite difference methods, resulted in a large system of equations which is usually solved iteratively.

Assume equation (1) as our model problem defined in a unit square Ω^h with spacings $\Delta x = \Delta y = \frac{1}{n} = h$ in both x and y directions with $x_i = x_0 + ih$ and $y_j = y_0 + jh$ for all i, j = 0,1,2,...., n. when equation (1) is discretized using finite difference scheme , the most commonly used approximations is the standard five-points stencil given by

$$v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{i,j} = h^2 f_{i,j}, \tag{2}$$

where $v_{i,j}$ is an approximation to the exact solution $u(x_i,y_j)$ at the point (x_i, y_j) = (ih, jh). Equation (1) can also be discretized using similar scheme with a width of 2h and leads to the following stencil,

$$v_{i+2,j} + v_{i-2,j} + v_{i,j+2} + v_{i,j-2} - 4v_{i,j} = 4h^2 f_{i,j}. \quad (3)$$

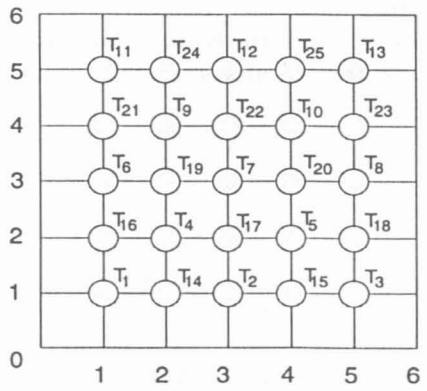

Fig 1. The solution domain Ω^h with the chessboard (CB) ordering strategy for o type of points

Another type of approximation derived from the rotated finite difference approximation can be obtained by rotating the x – y axis clockwise by 45°. Thus, the rotated difference approximation for equation (1) become (Abdullah (1991)),

$$v_{i+1,j+1} + v_{i-1,j-1} + v_{i+1,j-1} + v_{i-1,j+1} - 4v_{i,j} = 2h^2 f_{i,j}. \quad (4)$$

Equations (2), (3) and (4) have been used in the derivation of the new point iterative algorithm. A brief description of the full- and half-sweep points iterative algorithms are given below.

FULL-SWEEP POINT ITERATIVE ALGORITHM

Let us consider the solution at any point Ω^h may be obtained using the stencil five points finite difference approximation (equation (2)). The SOR algorithm involves may be described as follows

1. Define all the points in Ω^h, see Figure 1. Compute the value of h^2 beforehand and assign to a variable H.
2. Implement the relaxation procedure,

$$v_{i,j}^{(k+1)} = \omega\left(\tilde{v}_{i,j}^{(k+1)} - v_{i,j}^{(k)}\right) + v_{i,j}^{(k)}$$

where the $\tilde{v}_{i,j}^{(k+1)}$ are the intermediate solutions of the (k + 1) th Gauβ-Seidel iteration defined by

$$\tilde{v}_{i,j}^{(k+1)} = 0.25*(v_{i,j+1}^{(k)} + v_{i-1,j}^{(k+1)} + v_{i+1,j}^{(k)} + v_{i,j-1}^{(k+1)} - H*f_{i,j}),$$

for all the points.

3. Check for convergence. If the iterative process converges, go to step (4), otherwise, repeat the iteration cycle (i.e. go to step (2)).
4. Stop.

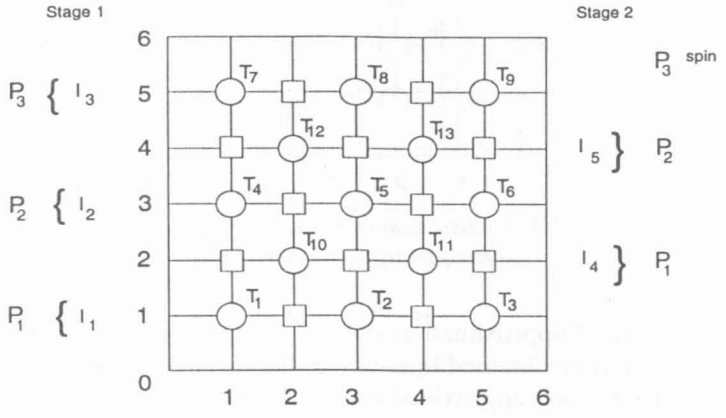

Fig 2. The solution domain Ω^h with the horizontal zebra line (HZL) ordering strategy for o type of points

HALF-SWEEP POINT ITERATIVE ALGORITHM

In this algorithm, the Ω^h is labelled into two types of points; ○ and □ as shown in Figure 2. The solution of any point either ○ or □ can be implemented by only involving the same type of point.

Using equation (4) and taking any group of two points (i.e. ○ and □) in Ω^h leads to the (2 x 2) system of equation,

$$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} v_{i,j} \\ v_{i+1,j} \end{bmatrix} = \begin{bmatrix} v_{i-1,j-1} + v_{i+1,j-1} + v_{i-1,j+1} + v_{i+1,j+1} - 2h^2 f_{i,j} \\ v_{i,j-1} + v_{i,j+1} + v_{i+2,j-1} + v_{i+2,j+1} - 2h^2 f_{i+1,j} \end{bmatrix}. \quad (5)$$

Splitting equation (5) leads to a decoupled group of (1 x 1) equations in explicit form as,

$$\left[v_{i,j}\right]=\frac{1}{4}\left[v_{i-1,j-1}+v_{i+1,j-1}+v_{i-1,j+1}+v_{i+1,j+1}-2h^2 f_{i,j,}\right] \quad (6)$$

and

$$\left[v_{i+1,j}\right]=\frac{1}{4}\left[v_{i,j-1}+v_{i,j+1}+v_{i+2,j-1}+v_{i+2,j+1}-2h^2 f_{i+1,j}\right]. \quad (7)$$

It is clear that equations (6) and (7) can be implemented by only involving points of type ○ and □, respectively. Therefore, the implementation of these equations can be carried out independently and the execution time can be reduced to nearly half if the iteration is carried out on either type of point. Hence, we may now define the half-sweep point iterative algorithm as,

1. Divide the Ω^h into two types of point; ○ and □, see Figure 2. Compute the values of h^2 and $2h^2$ beforehand and assign to variables H and I, respectively.
2. Implement the relaxation procedure,

   $$v_{i,j}^{(k+1)}=\omega\left(\tilde{v}_{i,j}^{(k+1)}-v_{i,j}^{(k)}\right)+v_{i,j}^{(k)}$$

   where the $\tilde{v}_{i,j}^{(k+1)}$ are the intermediate solutions of the (k + 1) th Gauβ-Seidel iteration defined by

   $$\tilde{v}_{i,j}^{(k+1)}=0.25*\left(v_{i-1,j+1}^{(k)}+v_{i-1,j-1}^{(k+1)}+v_{i+1,j+1}^{(k)}+v_{i+1,j-1}^{(k+1)}-I*f_{i,j}\right),$$

   for all the ○ points.
3. Check for convergence. If the iterative process converges, evaluate the solutions at the other half of points (i.e. □) using equation,

   $$v_{i,j}=0.25*\left(v_{i+1,j}+v_{i-1,j}+v_{i,j-1}+v_{i,j+1}-H*f_{i,j}\right),$$

   otherwise, repeat the iteration cycle (i. e. go to step (2)).
4. Stop.

QUARTER - SWEEP POINT ITERATIVE ALGORITHM

The Ω^h is labelled in three different types of points; ●, □ and ○ as shown in Figure 3. A group of ● points which involved in the iterative evaluations is about a quarter of the total point for a large size of points. The solution of any ● point can be computed by only involving points of type ●. Therefore, this computation can be carried out independently from the other two types of points; □ and ○.

Due to this independency, we can theoretically save the execution time by approximately a quarter if the iteration over the Ω^h is carried out only on the ● type of points. After the convergence criteria is achieved, the solutions of the remaining two types of points are executed directly at once starting from point type ○ and followed by □ using the equations (4) and (2), respectively. Hence , we may define the quarter-sweep point iterative algorithm as follows:

1. Divide the Ω^h into three types of point; ●, ○ and □, see Figure 3. Compute the values of h^2, $2h^2$ and $4h^2$ beforehand and assign to variables H, I and J, respectively.
2. Implement the relaxation procedure,

$$v_{i,j}^{(k+1)} = \omega\left(\tilde{v}_{i,j}^{(k+1)} - v_{i,j}^{(k)}\right) + v_{i,j}^{(k)}$$

where the $\tilde{v}_{i,j}^{(k+1)}$ are the intermediate solution of the (k + 1) th Gauβ-Seidel iteration defined by

$$\tilde{v}_{i,j}^{(k+1)} = 0.25 * (v_{i+2,j}^{(k)} + v_{i-2,j}^{(k+1)} + v_{i,j+2}^{(k)} + v_{i,j-2}^{(k+1)} - J * f_{i,j}),$$

for all the ● points.

3. Check for convergence. If the iterative process converges, evaluate the solutions at the other two points starting from point type ○ and followed by □ using the following

3.1. $v_{i,j} = 0.25 * (v_{i+1,j+1} + v_{i-1,j-1} + v_{i+1,j-1} + v_{i-1,j+1} - I * f_{i,j})$, and

3.2. $v_{i,j} = 0.25 * (v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - H * f_{i,j})$,

respectively. Otherwise, repeat the iteration cycle (i. e. go to step (2)).

4. Stop.

The details of the algorithm can be found in Othman *et al.* (1998).

PARALLEL IMPLEMENTATION AND STRATEGIES

Assume the Ω^h is large with even size of points n. The optimal parallel strategy of parallelizing the full-, half- and quarter- sweep point iterative algorithms have been investigated and can be outlined as follows:

For Quarter-sweep Point Iterative Algorithm: From Figure 3, each ● point or task T_i for all i = 1, 2,..., N with $N = \frac{1}{4}(n-2)^2$ is assigned to available processor one at a time in CB strategy. The static schedulling is employed in this

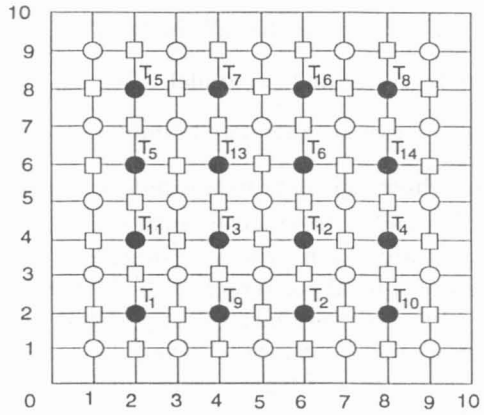

Fig 3. The solution domain Ω^h with the CB ordering strategy for ● type of points

implementation. By applying equation (3) in turn with such strategy to each task T_i in Ω^h will leads to a linear system as

$$\begin{bmatrix} D_r & U \\ U^T & D_b \end{bmatrix}\begin{bmatrix} u_r \\ u_b \end{bmatrix}=\begin{bmatrix} f_r \\ f_b \end{bmatrix} \tag{8}$$

with the diagonal sub matrices D_r and D_b of size $\left(\frac{N}{2}\times\frac{N}{2}\right)$ and each diagonal element is equivalent to -4. Applying the Gauβ-Seidel to equation (8), we will have

$$\begin{bmatrix} D_r & U \\ U^T & D_b \end{bmatrix}\begin{bmatrix} u_r \\ u_b \end{bmatrix}^{(k+1)} = \begin{bmatrix} f_r \\ f_b \end{bmatrix}-\begin{bmatrix} 0 & U \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_r \\ u_b \end{bmatrix}^{(k)}. \tag{9}$$

If the diagonal sub matrices D_r^{-1} and D_b^{-1} exist, we can evaluate equation (9) by first calculating

$$u_r^{(k+1)} = (1-\omega_e)u_r^{(k)} + \omega_e D_r^{-1}\left[f_r - Uu_b^{(k)}\right] \tag{10}$$

followed by

$$u_b^{(k+1)} = (1-\omega_e)u_b^{(k)} + \omega_e D_b^{-1}\left[f_r - U^T u_r^{(k+1)}\right] \tag{11}$$

with the relaxation factor, ω_e.

From equation (10), it is clear that all the tasks in $u_r^{(k+1)}$ are independent of each other and can be computed in parallel. After $u_r^{(k+1)}$ has been completed, equation (11) or $u_b^{(k+1)}$ can be calculated simultaneously using the updated values of $u_r^{(k+1)}$ since these calculations are independent. At the end of each stage, a synchronization call is executed to ensure the updated values are used in the subsequent iteration. Each processor independently iterate on its own task and check for its own local convergence. If the local convergence criteria is achieved then the global convergence test is performed; otherwise repeat the iteration cycle. If the global convergence criteria is achieved, the solutions at the remaining tasks (i.e. points of types ○ and □) will be evaluated directly at once, starting from points type ○ and followed by □ using equations (4) and (2), respectively. These direct evaluations can be executed in parallel by assigning each remaining task in natural strategy to available processor one at a time. Otherwise increase the number of iteration and repeat the iteration cycle.

For Half-sweep Point Iterative Algorithm: The algorithm uses the horizontal zebra line (HZL) strategy which consists of two stages of calculation upon each iteration, see Figure 2. At the first stage, three rows of the ○ points or tasks T_i are assigned to each processor in alternate order and the same happen in stage two. Lines l_1, l_2 and l_3 are assigned to P_1, P_2 and P_3, respectively. After calculating all values at the half of the tasks in given lines in parallel, a synchronization call takes place which marks the end of the first stage. In the second stages, lines l_4 and l_5 are assigned to P_1 and P_2 respectively while P_3 is keep on spinning. In other words, a group of lines l_1, l_2, l_3 is updated at stage 1 and then followed by a second group of lines l_4, l_5 which is updated in stage 2. By applying equation (4) in turn to each group with such strategy, we will have a large linear system

$$\begin{bmatrix} D_{l_1} & & & U & \\ & D_{l_2} & & U & U \\ & & D_{l_3} & & U \\ U^T & U^T & & D_{l_4} & \\ & U^T & U^T & & D_{l_5} \end{bmatrix} \begin{bmatrix} u_{l_1} \\ u_{l_2} \\ u_{l_3} \\ u_{l_4} \\ u_{l_5} \end{bmatrix} = \begin{bmatrix} f_{l_1} \\ f_{l_2} \\ f_{l_3} \\ f_{l_4} \\ f_{l_5} \end{bmatrix} \qquad (12)$$

where the diagonal sub matrices D_{l_1}, D_{l_2}, D_{l_3} and D_{l_4}, D_{l_5} of size $\left(\left\lceil\frac{N}{2}\right\rceil \times \left\lceil\frac{N}{2}\right\rceil\right)$ and $\left(\left\lfloor\frac{N}{2}\right\rfloor \times \left\lfloor\frac{N}{2}\right\rfloor\right)$, respectively with N = (n - 1) and the diagonal element is equivalent to –4. Since there are two stages, equation (12) can be rewrite as the following form

$$\begin{bmatrix} \hat{D}_1 & C \\ C^T & \hat{D}_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \tag{13}$$

By applying the Gauβ-Seidel to equation (13), we will have the following equation

$$\begin{bmatrix} \hat{D}_1 & 0 \\ C^T & \hat{D}_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^{(k+1)} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} - \begin{bmatrix} 0 & C \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^{(k)}. \tag{14}$$

The explicit solution of equation (14) can be de-coupled into the following system of equations

$$U_1^{(k+1)} = (1-\omega_e)U_1^{(k)} + \omega_e \hat{D}_1^{-1}\left[B_1 - CU_2^{(k)}\right] \tag{15}$$

and

$$U_2^{(k+1)} = (1-\omega_e)U_2^{(k)} + \omega_e \hat{D}_2^{-1}\left[B_2 - C^T U_1^{(k+1)}\right] \tag{14}$$

with the diagonal sub matrices $\hat{D}_1$ and $\hat{D}_2$ exist.

Clearly, we can see that all the tasks in $U_1^{(k+1)}$ are independent of each other and can be computed in parallel. Each processor is assigned an approximately equal number of tasks to work on. After $U_1^{(k+1)}$ has been calculated, $U_2^{(k+1)}$ can be calculated simultaneously using the updated values of $U_1^{(k+1)}$ since this calculation is independent. However, since the most recent values of $U_1^{(k+1)}$ are to be used in equation (16), a synchronizing call has to be made before the calculation of $U_2^{(k+1)}$ strats. Each processor then checks for its local and global convergence criteria the same way as described in the previous method. Once the global convergence is achieved, the solution at the remaining tasks (i. e. points of type □) will be evaluated directly in parallel at once using equation (4) by assigning tasks of each row to different processor.

For Full-sweep Point Iterative Algorithm: All ○ points or tasks T_i for all i = 1, 2, ..., N with N = $(n-1)^2$ which involve in the process of iterative evaluation are assigned to available processor one at a time in the CB strategy, see Figure 1. This strategy is the same as mentioned in the quarter-sweep point iterative algorithm. If the local and global convergence criteria are achieved, the

TABLE 1
Relaxation factor ω_e, no. of iteration, strategies and max. error for all the parallel algorithms

h^{-1}	Method	ω_e	No. iteration	Strategies	Max. error
	Full-	1.77	96	CB	5.42×10^{-6}
24	Half-	1.70	69	HZL	2.88×10^{-4}
	Quarter-	1.59-1.60	49	CB	2.64×10^{-5}
	Full-	1.84	145	CB	2.42×10^{-6}
36	Half-	1.76	103	HZL	1.28×10^{-4}
	Quarter-	1.71	73	CB	1.06×10^{-5}
	Full-	1.89	203	CB	1.25×10^{-6}
50	Half-	1.84	141	HZL	6.64×10^{-4}
	Quarter-	1.78	98	CB	5.28×10^{-6}
	Full-	1.92	281	CB	6.24×10^{-7}
70	Half-	1.89	205	HZL	3.38×10^{-5}
	Quarter-	1.84	136	CB	2.63×10^{-6}
	Full-	1.94	380	CB	3.15×10^{-7}
100	Half-	1.92	289	HZL	1.66×10^{-5}
	Quarter-	1.89	203	CB	1.27×10^{-6}

iterative evaluation is stopped; otherwise repeat the iteration cycle.

PERFORMANCE EVALUATION

In order to confirm that the parallel quarter-sweep point iterative algorithm is better than the other parallel algorithms, the following experiments were carried out on the SMP parallel computer, the Sequent S27. All algorithms were applied to the following test problem,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = (x^2 + y^2)e^{xy}, \quad (x,y) \in \Omega^h = [0,1] \times [0.1], \tag{17}$$

subject to the Dirichlet conditions and satisfying the exact solution $u(x,y) = e^{xy}$ for $(x,y) \in \partial\Omega^h$.

Throughout the experiments, the local convergence test was the maximum absolute error with the error tolerance $\varepsilon = 10^{-10}$. The experiments were performed on various sizes of n such as 24, 36, 50, 70, 100 and number of processors ranging from 1 to 5. For each n, the experimental value of ω_e was obtained to within ±0.01 by solving the problem for a range of values of ω_e and choosing those which give the minimum number of iterations. Table 1 lists the optimum value of ω_e number of iterations, strategies and maximum error for all the parallel algorithms and the timing results and speedup are presented in Table 2. For n = 100, the graphs of execution time, speedup and efficiency

TABLE 2
Execution time and speedup for all the parallel algorithms

h^{-1}	No. processors	Full-		Half-		Quarter-	
		Time	Speedup	Time	Speedup	Time	Speedup
24	1	4.8519	1.0000	1.8878	1.0000	0.6441	1.0000
	2	2.6289	1.8456	1.0521	1.7943	0.3739	1.7225
	3	1.8714	2.5926	0.8021	2.3533	0.2828	2.2770
	4	1.4975	3.2400	0.6100	3.0947	0.2211	2.9131
	5	1.3322	3.6420	0.5623	3.3572	0.2002	3.2164
36	1	16.6563	1.0000	6.3132	1.0000	2.1570	1.0000
	2	8.8748	1.8768	3.3785	1.8686	1.1685	1.8465
	3	6.4233	2.5931	2.5405	2.4850	0.8974	2.4036
	4	5.0936	3.2681	2.0121	3.1376	0.6905	3.1237
	5	4.2167	3.9500	1.6748	3.7695	0.5970	3.6125
50	1	45.5404	1.0000	16.6219	1.0000	5.5789	1.0000
	2	24.0318	1.8950	8.8426	1.8797	3.0786	1.8121
	3	16.8674	2.6999	6.4321	2.5842	2.2586	2.4700
	4	13.6692	3.3316	5.0450	3.2947	1.7155	3.2520
	5	10.9319	4.1658	4.1124	4.0418	1.4101	3.9562
70	1	125.8978	1.0000	48.3252	1.0000	15.3360	1.0000
	2	67.8831	1.9000	25.5685	1.8900	8.3356	1.8398
	3	46.2469	2.7889	17.7666	2.7200	5.8891	2.6041
	4	34.6940	3.6288	13.7592	3.5122	4.5747	3.3523
	5	28.6456	4.3950	11.4713	4.2127	3.8009	4.0348
100	1	353.5294	1.0000	146.2460	1.0000	48.1291	1.0000
	2	179.8033	1.9662	76.9520	1.9004	25.3738	1.8968
	3	127.3153	2.7768	52.8097	2.7693	17.6200	2.7315
	4	96.1932	3.6752	41.4294	3.5300	13.7472	3.5010
	5	79.1159	4.4685	33.8454	4.3210	11.5533	4.1658

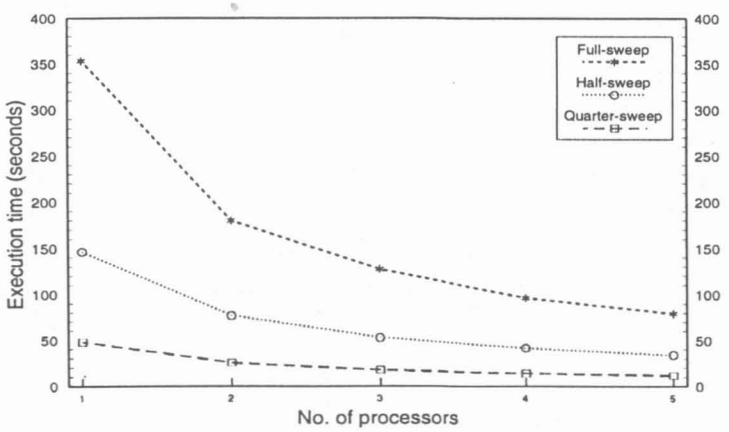

Fig 4. Execution time versus number of processors for n=100

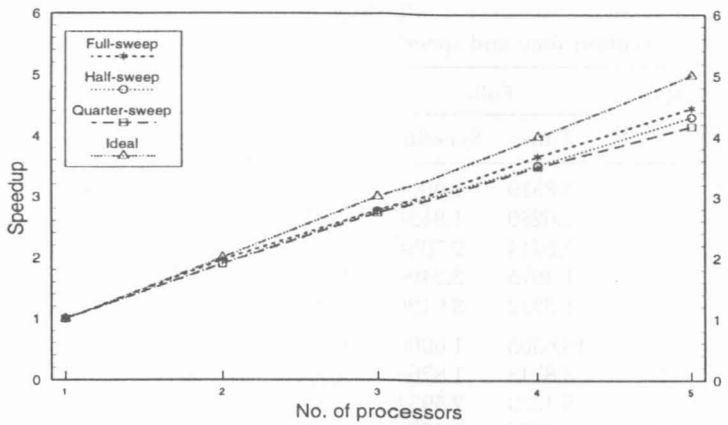

Fig 5. Speedup versus number of processors for n=100

versus number of processors are shown in *Figures 4, 5* and *6,* respectively.

The temporal performance parameter is usually used to compare the performance of different algorithms for solving the similar problem. It is defined as the inverse of the execution time where the unit is work done per second. The algorithm with the highest performance executes in the least time and therefore is the better algorithm. The graph of temporal performance versus number of processors of all the parallel algorithms is plotted and shown in Figure 7.

CONCLUSION

In Table 2, the timing results obtained have shown that the parallel quarter-

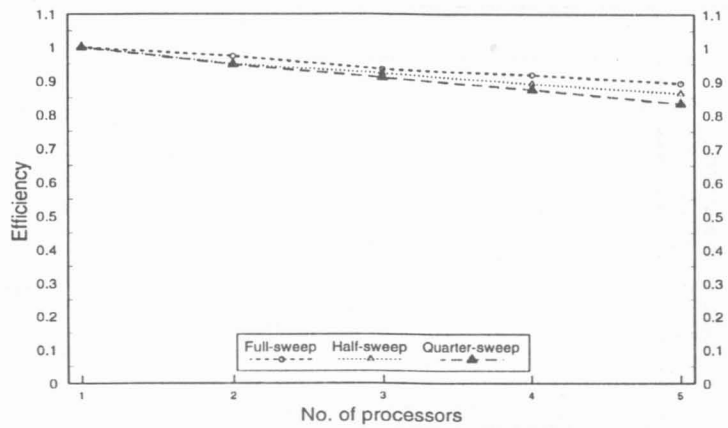

Fig 6. Efficiency versus number of processors for n=100

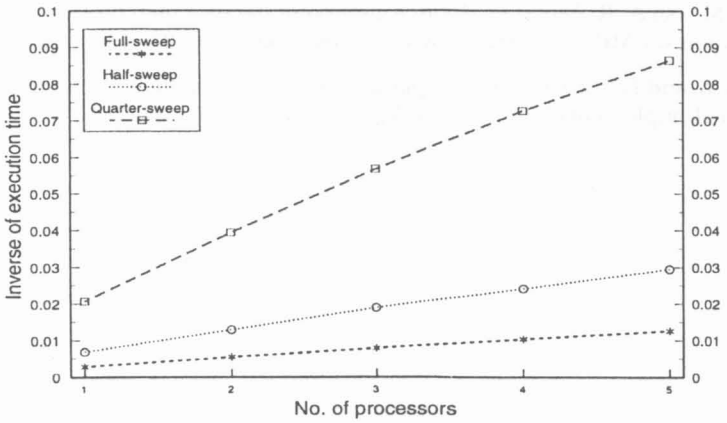

Fig 7. Temporal performance versus number of processors for n=100

sweep point iterative algorithm which uses the CB strategy is superior than the parallel full- and half-sweep point iterative algorithms for any number of processors and as n gets larger. Figure 4 shows the graph of the execution time versus number of processors for case n = 100. This is due to the lower total computational operations in the algorithm since approximately a quarter of the total points are involved in the iteration. The superiority of the algorithm is also indicated by the highest value of the temporal performance and least number of iteration of the algorithm as shown in Figure 7 and Table 1, respectively.

However, the speedup and efficiency of the parallel quarter-sweep algorithm is slightly less than the other two algorithms and it can be improved by increasing the size of points n in the Ω^h, see Figures 5 and 6. In conclusion, the parallel quarter-sweep point iterative algorithm with the CB strategy performs drastic improvement in execution time and it has proved to be an efficient parallel Poisson algorithm among the three algorithms on the SMP parallel computer.

REFERENCES

ALI, N. M. and A. R. ABDULLAH. 1997. New Parallel Point Iterative Solutions for the Diffusion Convection Equation, in *Proc. of the IASTED Intern. Conf. on Parallel and Distributed Computing and Networks,* ed. M. H. Hamza, pp. 136-139, IASTED-Acta Press, Zürich.

ABDULLAH, A. R. 1991. The four points explicit decoupled group (EDG) method: a fast Poisson solver. *Intern. Journal of Computers and Mathematics* **38** : 61-70.

BARLOW, R. H and D. J. EVANS. 1982. Parallel algorithms for the iterative solution to linear system. *Computer Journal* **25** (1): 56-60.

EVANS, D. J. 1984. Parallel S.O.R. iterative methods. *Parallel Computing* **1**: 3-18.

OTHMAN, M. AND A. R. ABDULLAH. 1998. A new point iterative method for solving Poisson equation on MIMD computer system. *Sains Malaysiana* **27** (1&2): (in press).

YOUSIF, W. S. and D. J. EVANS. 1995. Explicit de-coupled group iterative methods and their parallel implementations. *Parallel Algorithms and Applications* **7**: 53-71.