LRP 499/94 September 1994

A DIRECT POISSON SOLVER FOR PARTICLE-IN-CELL (PIC) SIMULATION

T.M. Tran, K. Appert and O. Sauter

A direct Poisson solver for Particle-In-Cell (PIC) simulation

T.M. Tran, K. Appert and O. Sauter
CRPP, Association Euratom-Confédération Suisse,
Ecole Polytechnique Fédérale de
Lausanne, CH-1007 Lausanne, Switzerland

ABSTRACT

A direct Poisson solver, based on the isoparametric finite element discretization and a domain decomposition technique, is described. A simple parallelization scheme is proposed and evaluated on a 128 processor Cray T3D.

1. Introduction

In particle-in-cell (PIC) simulations [1–2], there are two distinct parts: (1) A particle pushing part, using Newton's equation of motion and (2) a second part where the electromagnetic fields created by the particles (through their charges and currents) should be found from the Maxwell's equations. In the electrostatic approximation in which the high-frequency magnetic field is neglected, only Poisson's equation for the electric field has to be solved.

Rapid solvers such as the fast Fourier transform [3] and the cyclic reduction [4] based methods are generally utilized to solve the Poisson's in PIC simulations. They are, however, restricted to rather simple geometries. The solution method described in this report is specially developed to be applied to general 2D curvilinear geometries and arbitrary shapes. The isoparametric bilinear finite elements [5] are chosen to discretize Poisson's equation on a non-rectangular mesh (section 2). Using the domain decomposition technique, complicated shapes of the computational domain can be handled, resulting in the linear system $Ax = b$, where the matrix A has a block structure. The two-step (factorization and solve) direct method to solve this system together with a simple scheme to parallelize the algorithm are presented in section 3. It is important to note that only the solve step is required in the time loop of the PIC simulations since A is constant. The timing performances of the parallel solver on a 128 processors Cray T3D system are shown in section 4. Section 5 contains the concluding remarks.

2. Discretization of the Poisson's equation

Consider Poisson's equation in 2D cylindrical (r, z) coordinates

$$\frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial \phi}{\partial r} + \frac{\partial^2 \phi}{\partial z^2} = -\frac{\rho}{\epsilon_0}, \qquad \text{in } \Omega. \tag{1}$$

This differential equation, with given boundary conditions, can be shown to be equivalent to the following variational formulation [5]: Find ϕ such that

$$\iint_\Omega \left(\frac{\partial \phi}{\partial r}\frac{\partial v}{\partial r} + \frac{\partial \phi}{\partial z}\frac{\partial v}{\partial z} \right) r dr dz = \iint_\Omega \frac{\rho v}{\epsilon_0} r dr dz, \qquad \text{for every } v. \tag{2}$$

Assume that the computational domain Ω is subdivided into quadrilateral cells which may be non-rectangular in order to fit the curved boundaries. The integrals in Eq. (2) can then be performed using the isoparametric transformation [5] defined as

$$\begin{aligned} r(\xi,\eta) &= \alpha_1 + \alpha_2\xi + \alpha_3\eta + \alpha_4\xi\eta, \\ z(\xi,\eta) &= \beta_1 + \beta_2\xi + \beta_3\eta + \beta_4\xi\eta, \end{aligned} \tag{3}$$

which maps quadrilaterals on the plane (r, z) to squares on the (ξ, η) plane. Using the local node ordering shown in Fig. 1, the coefficients (α_i, β_i) can be computed as

$$\begin{aligned} \alpha_1 &= (r_1 + r_2 + r_3 + r_4)/4 & \beta_1 &= (z_1 + z_2 + z_3 + z_4)/4 \\ \alpha_2 &= (-r_1 + r_2 - r_3 + r_4)/4 & \beta_2 &= (-z_1 + z_2 - z_3 + z_4)/4 \\ \alpha_3 &= (-r_1 - r_2 + r_3 + r_4)/4 & \beta_3 &= (-z_1 - z_2 + z_3 + z_4)/4 \\ \alpha_4 &= (r_1 - r_2 - r_3 + r_4)/4 & \beta_4 &= (z_1 - z_2 - z_3 + z_4)/4 \end{aligned} \tag{4}$$

The contribution from a given cell to the integral on the left-hand-side of Eq. (2) can be approximated by choosing the lowest order bilinear test functions defined on the square $-1 \le \xi \le +1$, $-1 \le \eta \le +1$:

$$v(\xi,\eta) = \Lambda_i(\xi)\Lambda_j(\eta), \quad i = 1,2,\ j = 1,2 \tag{5}$$

where

$$\begin{aligned} \Lambda_1(x) &= (1-x)/2, \\ \Lambda_2(x) &= (1+x)/2, \end{aligned}$$

and expanding the unknown function $\phi(\xi, \eta)$ in terms of the same bilinear functions

$$\phi(\xi,\eta) = \phi_1\Lambda_1(\xi)\Lambda_1(\eta) + \phi_2\Lambda_2(\xi)\Lambda_1(\eta) + \phi_3\Lambda_1(\xi)\Lambda_2(\eta) + \phi_4\Lambda_2(\xi)\Lambda_2(\eta) \tag{6}$$

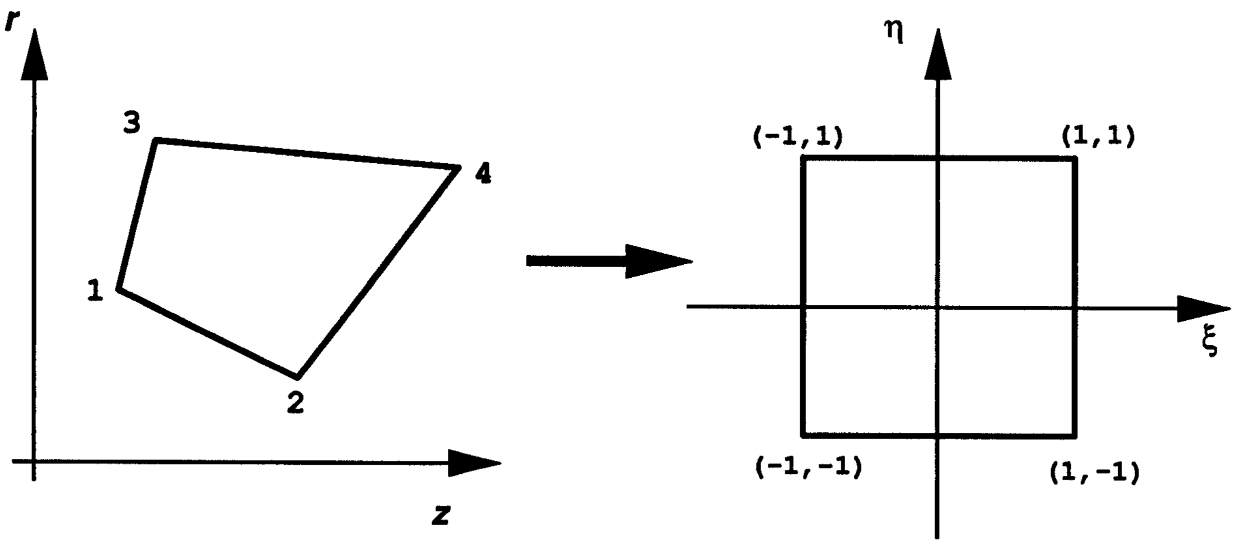

Figure 1: The isoparametric transformation $(r,z) \rightarrow (\eta,\xi)$. The 4 corners of the quadrilateral cell 1, 2, 3 and 4 are respectively mapped to the points $(-1,-1)$, $(1,-1)$, $(-1,1)$ and $(1,1)$.

where $(\phi_1, \phi_2, \phi_3, \phi_4)$ are the finite element solution for the potential ϕ on the four nodes of the cell. The derivatives are computed, using the chain rule and Eq. (3):

$$\begin{aligned}\frac{\partial\phi}{\partial\xi} &= (\alpha_2+\alpha_4\eta)\frac{\partial\phi}{\partial r}+(\beta_2+\beta_4\eta)\frac{\partial\phi}{\partial z},\\ \frac{\partial\phi}{\partial\eta} &= (\alpha_3+\alpha_4\xi)\frac{\partial\phi}{\partial r}+(\beta_3+\beta_4\xi)\frac{\partial\phi}{\partial z}.\end{aligned} \tag{7}$$

The right-hand-side of Eq. (2) is obtained by noting that the charge density ρ can be written in terms of the particle coordinates $(r_p, z_p), p = 1, \ldots, N_p$ as

$$\rho(r,z) = \sum_{p=1}^{N_p} q_p \frac{1}{r}\delta(r-r_p)\,\delta(z-z_p)$$

and thus

$$\iint_\Omega \rho v\, r dr dz = \sum_{p=1}^{N_p} q_p v\left[\xi(r_p,z_p), \eta(r_p,z_p)\right] \tag{8}$$

which simply reduces to the standard CIC charge assignment [1] when the cell in (r,z) is rectangular.

Repeating the procedure of calculating the integrals just described, for every cells, a linear system of equations for the potentials ϕ defined on the mesh nodes is obtained. The electric field $\vec{E} = -\nabla\phi$ on the particle position, needed in the particle pushing, will be computed locally in the following way.

- Obtain the particle coordinates in the isoparametric space (ξ_p, η_p) from (r_p, z_p) by inverting the Eqs. (3).
- Interpolate the local electric field

$$(E_\xi, E_\eta) = -(\partial\phi/\partial\xi, \partial\phi/\partial\eta)$$

  on the particle using the prescription that $\partial\phi/\partial\xi$ $(\partial\phi/\partial\eta)$ is piecewise linear (constant) in η and piecewise constant (linear) in ξ, consistently with the approximation chosen in Eq. (6).
- Finally invert Eqs. (7) to obtain the field components (E_r, E_z).

Note that this way of interpolating the field on the particle produces discontinuities for the electric field as the particle crosses the cell boundaries. It has however, the advantage that the scheme is completely local and can be easily parallelized.

3. The solver algorithm

The isoparametric finite element discretization of the Poisson's equation, described in the previous section leads to the problem of solving $Ax = b$ where x is the vector of the unknown potentials ϕ on the mesh nodes, b is the known source vector and A is a symmetric positive definite matrix. When the computational domain Ω is a single quadrilateral of $N = N_1 \times N_2$ cells (where N_1 and N_2 are the number of intervals in the directions z and r respectively), A is a band matrix. By ordering the elements first in the direction of increasing z for example, the half bandwidth of A is simply $m = N_1 + 2$. In that case, the linear system can be solved by computing first the Cholesky factorization

$$A = LL^T \tag{9}$$

where L is a lower triangular matrix with the same half-bandwidth m. The solution vector x can then be obtained by successive forward and back substitutions to solve the triangular systems

$$Ly = b, \quad L^T x = y. \tag{10}$$

The factorization and solve steps could be done by calling the `LAPACK` [7] routines `SPBTRF` and `SPBTRS`. One can show that the number of floating-points operations required for the factorization is approximately m (m being the matrix half-bandwidth) times larger than that required for the solve step.

The alternative is to decompose the computational domain Ω into N_s smaller quadrilateral sub-domains. With this technique, more complex shapes of Ω could be considered. On the other hand, we will also use this decomposition method, even in the case of a simple rectangular domain for parallelization.

By numbering the interior nodes of every sub-domains before the points (which we will refer to as connectivity points) on the boundary between a pair of sub-domains, the matrix A acquires the following block structure:

$$A = \begin{pmatrix} A_1 & & 0 & V_1 \\ & \ddots & & \vdots \\ 0 & & A_{N_s} & V_{N_s} \\ V_1^T & \dots & V_{N_s}^T & D \end{pmatrix}, \tag{11}$$

where the diagonal matrices A_k are band symmetric positive definite, the off-diagonal matrices V_k are sparse and the connectivity matrix D is symmetric positive definite. Writing the solution vector $x = (x_1, \dots, x_{N_s}, x_d)^T$ and the right-hand-side vector $b = (b_1, \dots, b_{N_s}, b_d)^T$, the linear system could be expressed in a more explicit form as

$$\begin{aligned} A_k x_k + V_k x_d &= b_k, \quad k = 1, \dots, N_s \\ \sum_{k=1}^{N_s} V_k^T x_k + D x_d &= b_d \end{aligned}$$

which can be reduced to

$$A_k x_k = b_k - V_k x_d, \quad k = 1, \dots, N_s \tag{12a}$$

$$\bar{D} x_d = b_d - \sum_{k=1}^{N_s} V_k^T A_k^{-1} b_k \tag{12b}$$

where

$$\bar{D} = D - \sum_{k=1}^{N_s} V_k^T A_k^{-1} V_k. \tag{13}$$

The factorization algorithm (known as the *asymmetric block factorization* in the literature [6]) can be described as follows, together with the sequence of calls to the LAPACK routines:

1. Initialize $\bar{D} \leftarrow D$

2. For each $k = 1, \dots, N_s$

```
        2.1. Factor $A_k = L_k L_k^T$    (SPBTRF)
        2.2. Solve $(L_k L_k^T)W = V_k$    (SPBTRS)
        2.3. Modify $\bar{D} \leftarrow \bar{D} - V_k^T W$
    3. Factor $\bar{D} = L_{\bar{D}} L_{\bar{D}}^T$    (SPOTRF)
```

We have introduced a temporary matrix W in the step 2.2. To avoid defining a whole matrix, one can solve (step 2.2) and compute $\bar{D}$ (step 2.3), column by column. In addition to save memory storage, it is possible to reduce the number of operations since V_k is sparse and may have many null columns. At the end of the factorization, only the matrices L_k, V_k and $L_{\bar{D}}$ need to be kept, from which the solution of the linear system is straightforward:

```
    1. For each $k = 1, \ldots, N_s$
        1.1. Solve $(L_k L_k^T)y_k = b_k$    (SPBTRS)
        1.2. Compute $b_d \leftarrow b_d - V_k^T y_k$
    2. Solve $(L_{\bar{D}} L_{\bar{D}}^T)x_d = b_d$    (SPBTRS)
    3. For each $k = 1, \ldots, N_s$
        3.1. Solve $(L_k L_k^T)t = V_k x_d$    (SPBTRS)
        3.2. Compute $x_k = y_k - t$
```

Here, t is a temporary array. During the solve step, b_k is overwritten by y_k (step 1.1) and then by x_k (step 3.2) while b_d is overwritten by x_d (step 2), since the right-hand-side array b usually is no more required.

In the parallel implementation, we consider only the solve phase since the factorization (which could be performed on a serial machine) needs to be done only once. In addition to the factorization just described above, the inverse of the connectivity matrix $\bar{D}$ is computed by calling for example the LAPACK routine SPOTRI. The data partition among the N_s processors can then be done in the following way. The sub-matrices (L_k, V_k) and the partial right-hand-side vector b_k are assigned to processor k $(k = 1, \ldots, N_s)$. The $N_c \times N_c$ matrix $\bar{D}^{-1}$ (where N_c is the number of connectivity points) is subdivided into N_s rectangular sub-matrices $\bar{D}_k^{-1}$ of r_k contiguous rows and N_c columns and distributed to the N_s processors, with r_k given by

$$r_k = \begin{cases} \mathrm{int}\,(N_c/N_s) + 1, & \text{for } \mathrm{mod}(N_c, N_s) \text{ processors;} \\ \mathrm{int}\,(N_c/N_s), & \text{for the remaining processors.} \end{cases}$$

A copy of the full right-hand-side vector b_d is given to every processors. With this data partition, the parallel solve algorithm could be formulated for the processor k as follows.

```
1. Do in parallel
    1.1. Solve $(L_k L_k^T) y_k = b_k$    (SPBTRS)
    1.2. Compute $s_k \leftarrow V_k^T y_k$
2. Compute the sum reduction across every processors $s_k \leftarrow \sum s_k$
3. Do in parallel
    3.1. Compute $b_d \leftarrow b_d - s_k$
    3.2. Compute $x_d = \bar{D}_k^{-1} b_d$    (SGEMV)
4. Redistribute $x_d$
5. Do in parallel
    5.1. Solve $(L_k L_k^T) t = V_k x_d$    (SPBTRS)
    5.2. Compute $x_k = y_k - t$
```

Note that the steps 1, 3 and 5 are completely parallel. The step 2 involves communications across all the processors while the redistribution in step 4 involves only the few processors that need the elements of vector x_d to perform their back-substitution. The BLAS-2 routine SGEMV was invoked to perform the (rectangular) matrix-vector product in step 3.2. It is noteworthy that at the end of the solve, the solutions at the interior points *and* at the boundaries of a sub-domain k are available to the processor k so that the derivatives of the solutions (as defined in section 2) could be calculated locally.

4. Results on T3D

In the following, a rectangular domain of $N_1 \times N_2$ mesh cells is considered. We use a one-dimensional partition such that each sub-domain has equal size as shown in Fig. 2. This particular partition was chosen because in PIC simulation of beams propagating along z, the number of particles in these sub-domains is practically equal and constant in time, except during the start-up phase.

It is possible to estimate the size of the different matrices stored in each processor as follows. Denoting the half-bandwidth of L_k as $m_k \simeq N_1/N_s$ and the total number of connectivity points as $N_c = (N_2+1)(N_s-1)$, N_s being the number of sub-domains (processors), we have:

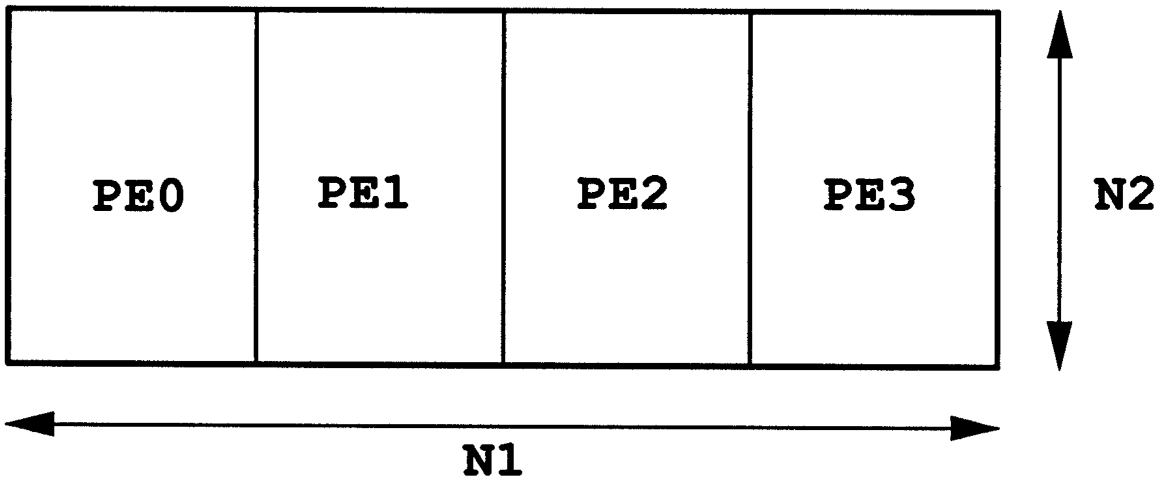

Figure 2: One-dimensional partition of a rectangular domain for a four processors case. The entire domain is meshed into $N_1 \times N_2$ cells.

- Local matrix L_k.

$$[L_k] \simeq m_k \frac{N_1}{N_s} N_2 \simeq \frac{N_1^2}{N_s^2} N_2 \tag{14a}$$

- Connectivity matrix $\bar{D}_k^{-1}$.

$$\left[\bar{D}_k^{-1}\right] \simeq \frac{N_c^2}{N_s} \simeq \frac{N_2^2 \left(N_s - 1\right)^2}{N_s} \simeq N_2^2 N_s \tag{14b}$$

while the number of non-zero elements of the sparse matrix V_k is negligible in most of the cases considered here.

The operation count per processor N_{op}, for the solve algorithm described in the previous section can then be estimated. The result,

$$N_{\text{op}} = 8\,[L_k] + 2\left[\bar{D}_k^{-1}\right] \simeq 2N_1N_2\left(\frac{4N_1}{N_s^2} + \frac{N_2N_s}{N_1}\right) \tag{15}$$

shows that the number of floating-point operations decreases almost as $1/N_s^2$ (N_s being the number of processors) down to a minimum at

$$N_s = N_{\text{crit}} \simeq \sqrt[3]{4N_1^2/N_2}, \tag{16}$$

where it starts to increase linearly with N_s. Hence, this method cannot be very efficient if used with a number of processors largely exceeding the critical number N_{crit}. On the other hand, it is very suitable for thin domains satisfying $N_1 \gg N_2$.

Elapsed times measured on the T3D are shown in Table 1. The time spent for the operation $x_d = \bar{D}_k^{-1} b_d$ (SGEMV) needed to solve for the connectivity points is indicated separately from the total time and labeled as "Conn. time". Apart from the data at

N. of Proc.	Total time (ms/step)	Conn. time (ms/step)
2	800.287	0.352
4	254.469	1.096
8	59.747	5.481
16	24.584	8.151
32	24.926	18.153
64	106.447	102.448
128	153.988	88.713

Table 1: Timing of the parallel solve algorithm for the 256$\times$64 problem. The total times and the times to solve for the connectivity points are obtained by taking an average over 100 executions of the solve loop.

$N_s = 64$, this connectivity time grows linearly as expected from the second term in Eq. (15). The overall behavior of the total time is roughly proportional to the numbers of operations, Eq. (15), and in particular, the minimum is well described by Eq. (16). The same is true for a 1024×140 case which is shown in Table 2. Note that it was not possible to run the latter case with a number of processors outside the range shown in the table due to the present memory limitation of the T3D processors (2 Mw). A summary of the results is depicted in Fig. 3.

Using the number of operations per processor $N_{\rm op}$ as given by Eq. (15) and Tables 1 and 2, the speed of computation (expressed in terms of Gflops) for the solver can be estimated. This is shown in Fig. 4 as a function of the number of processors. The solid line represents a speed of 10 Mflops per processor. Single-processor code optimization is expected to yield an improvement of this figure by a factor of 2 to 3. This effort, however, will be postponed until a final version of the solver is adopted.

N. of Proc.	Total time (ms/step)	Conn. time (ms/step)
16	638.406	32.740
32	190.902	72.122
64	190.131	153.014

Table 2: Timing of the parallel solve algorithm for the 1024 × 140 problem.

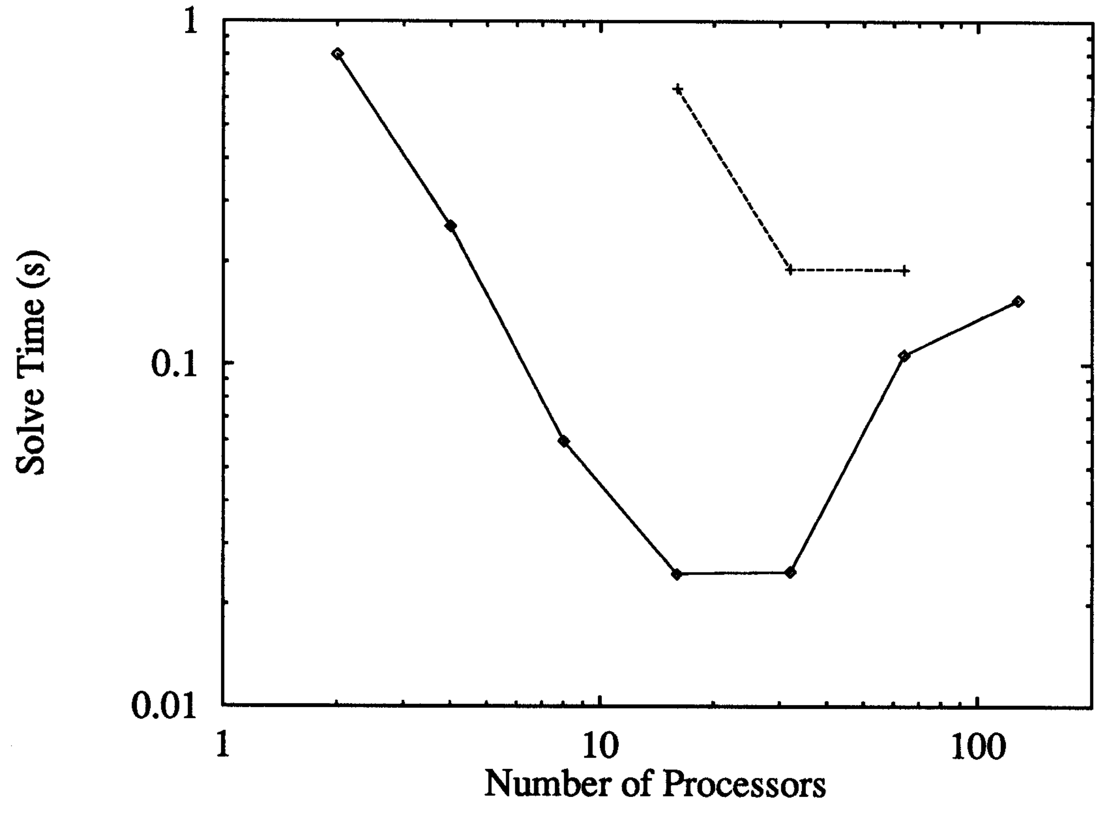

Figure 3: Solve time in seconds versus the number of processors on the T3D for the 256 × 64 problem (lower curve) and the 1024 × 140 problem (upper curve)

5. Conclusions

In this paper, we have presented a direct method to solve Poisson's equation for a 2D cylindrical geometry of arbitrary shapes. Only the parallelization of the solve step of the algorithm was considered and tested on a 128 processor Cray T3D system. It has been shown that this method is well suited for a moderate number of processing elements, Eq. (16), only; in our case, this number is of the order of 32 to 64, Fig. 3. The reason for the limitation is the inappropriate treatment of the connectivity matrix,

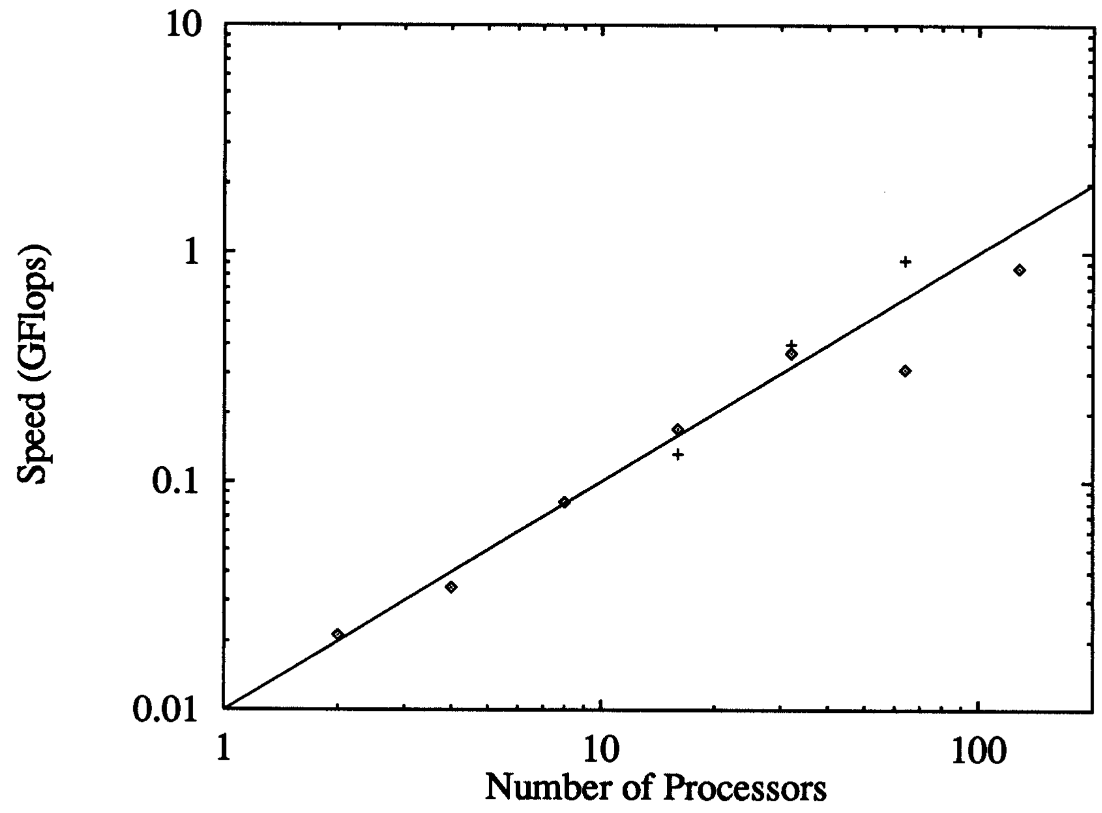

Figure 4: The computational speed in Gigaflops versus the number of processors on the T3D for the 256 × 64 case (◇) and the 1024 × 140 case (+). The solid line is given by a rate of 10 Mflops per processor.

Eq. (13), as a full matrix. An important improvement of the method could be achieved by noting that $\bar{D}$ is symmetric and has a band structure. In the benchmark problem considered, the half bandwidth is $m = 2N_2$ which implies that the size of $\bar{D}$ is

$$[\bar{D}] = mN_c = 2N_2N_c \simeq 2N_2^2N_s.$$

With a *scalable* solve routine (equivalent to the SPBTRS routine), we would expect that the time to solve the connectivity points is a constant function with respect to the number of processors, in contrast to the linear increase obtained when $\bar{D}$ is assumed to be full. Parallel methods to solve such a system have yet to be found.

So far, the factorization of A_k and the inverse of $\bar{D}$ have been obtained using the serial machine. A minor improvement of this code consists in making it less dependent on the serial machine. This implies the parallel factorization of $\bar{D}$ using for example SCALAPACK [8]. For this, however, a 2D secondary partition has to be introduced for $\bar{D}$ which may quite substantially increase the housekeeping.

Acknowledgments

This work was supported in part by the Swiss National Science Foundation and by the Cray Research Parallel Application Technology Program. It is a pleasure to thank the Cray personnel for their cooperative attitude, in particular we would like to thank Drs. K. Ghosh and M. Roche.

References

1. C.K. Birdsall, A.B. Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill Inc., 1985.
2. R.W. Hockney, J.W. Eastwood,*Computer Simulation using Particles*, Adam Hilger Inc., 1988.
3. R.W. Hockney, "A Fast Direct Solution of Poisson's Equation using Fourier Analysis", J. Assoc. Comput. Mach. 12, 95 (1965).
4. O. Buneman, "A Compact Non-Iterative Poisson-Solver", *SUIPR report* No. 294, Stanford University, 1969.
5. G. Strang, G. Fix, *An Analysis of the Finite Element Method*, 1973.
6. A. George, J.W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall Inc., 1981.
7. E. Anderson *et. al.*, *LAPACK Users' Guide*, SIAM, 1992..
8. J. Choi, J.J. Dongarra, D. Walker, R.C. Whaley, " SCALAPACK Reference Manual", Version 1.0Beta, ORNL/TM-12470, 1994.