LRP 528/95 August 1995

A DIRECT PARALLEL SPARSE MATRIX SOLVER

T.M. Tran, R. Gruber, K. Appert and S. Wuthrich

submitted for publication to
Comput. Phys. Commun.

A direct parallel sparse matrix solver

T.M. Tran[a], R. Gruber[a], K. Appert[a] and S. Wuthrich[b]

[a] CRPP, Association Euratom-Confédération Suisse, Ecole Polytechnique Fédérale de Lausanne, CH-1007 Lausanne, Switzerland

[b] Cray Research (Switzerland) S.A., CRAY-EPFL PATP Center, Parc Scientifique (PSE), CH-1015 Lausanne, Switzerland

ABSTRACT

The direct sparse matrix solver is based on a domain decomposition technique to achieve data and work parallelization. Geometries that have long and thin structures are specially efficiently tractable with this solver, provided that they can be decomposed mainly in one direction. Due to the separation of the algorithm into a factorization stage and a solution stage, time-dependent problems with a constant coefficient matrix are particularly well suited for this solver. The parallelization performances obtained on a Cray T3D show that the method scales up to at least 256 processors.

1. Introduction

The attraction of direct methods to solve linear sets of equations relies mainly on the separation of the computations into two steps: a factorization step and a solution step, the latter being an order of magnitude less expensive than the former. For linear problems with constant coefficient matrices and changing right-hand sides, the cost of the direct methods may be essentially the cost of the solution step, given the factorized matrices which are computed only once.

Parallel matrix solvers exist, such as the SCALAPACK library [1]. However, they are available presently only for dense matrices while those resulting from finite difference or finite element discretization of partial differential equations, are always sparse.

In this article, we present a parallel solver for such a sparse linear system of equations. The domain decomposition technique is used to distribute the data (matrices, solution and right-hand side vectors) as well as the computations among the processing elements. In order to handle a block tridiagonal matrix arising from the factorization stage, a parallel cyclic reduction is employed to minimize the computational and communication effort during the solution phase, resulting in a $\log_2 P$ scaling for the algorithm, where P is the number of processors.

The definition of the matrix problem is presented in section 2, together with the two-step algorithm for the matrix factorization and the backsolve step. The implementation of the solver is given in full detail in section 3. A rough estimation for the storage requirement and the operation count is also given in this section. The timing performances of the solver on a 256 processor Cray-T3D can be found in section 4 and finally, the section 5 contains some concluding remarks.

2. Matrix problem

2.1. The ordering strategy

The matrix solver we present is based on the old technique of decomposing a geometrical domain into sub-domains [2] in a structured way. This method permits the treatment of complicated shapes and the definition of the matrix partition for parallelization. We consider here only 2D domains and elliptic problems (such as the Poisson equation) discretized using bilinear finite elements [3]. Each sub-domain (which may have non-rectangular shapes) is meshed using quadrilaterals.

If the unknown variable associated with a given mesh node is related only to those on its immediate neighbors, the numbering of the nodes can be performed in the following order: (1) The nodes lying internal to a sub-domain, (2) the nodes lying on connectivity faces vertical to the major extension of the geometry and finally (3) the remaining horizontal connectivity nodes. An example of mesh and numbering for 8 sub-domains of 2×2 cells per sub-domain is shown in Fig. 1: The *vertical* and the *horizontal* connectivity nodes are shown here in thin border boxes and thick border boxes respectively. The nodes that are local to a given processor, are grouped inside boxes with dashed borders. In the chosen node partition, the horizontal connectivity nodes are duplicated on all processors, assuming that their number is small compared to the number of all other nodes. This is indeed the case for problems in which the domain decomposition can be performed mostly in one direction.

Using this ordering scheme, the matrix problem resulting from the original partial differential problem can be written as:

$$\begin{pmatrix} \mathcal{A} & \mathcal{V} & \mathcal{W} \\ \mathcal{V}^T & \mathcal{D} & \mathcal{Z} \\ \mathcal{W}^T & \mathcal{Z}^T & H \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} b \\ c \\ d \end{pmatrix}, \tag{1}$$

where the vectors (x, y, z) and (b, c, d) are respectively the unknowns and the right-hand-sides defined on the three types of nodes defined above. The sub-matrices $\mathcal{A}$, $\mathcal{D}$ and H, on the diagonal, relate the nodes of the same type and are symmetric. Note that $\mathcal{A}$ is block diagonal with blocks having a band structure, that $\mathcal{D}$ is also block diagonal but with blocks being tridiagonal and that H is simply a tridiagonal matrix. All of the off-diagonal sub-matrices $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{Z}$, which relate the nodes of different kinds, are sparse. A derivation of Eq. (1) from the discretization of the Poisson equation can be found in Ref. 4.

The overall structure of the matrix is illustrated in Fig. 2 for the example shown in Fig. 1. The structure of the sparse matrix $\mathcal{V}$ is made explicit, using the matrices U_k and V_k which act on the connectivity points on the left and the right side of a given sub-domain k, respectively. From the numbering scheme defined in Fig. 1 follows that D_6, V_6, Z_6 and U_7 are all empty blocks since the sub-domains 6 and 7 are deconnected: The unknown and right-hand-side vectors y_6 and c_6 are empty. One can then consider two decoupled linear systems for the connectivity nodes $y_1, y_2, \ldots, y_5$ and y_7 respectively.

In Fig. 2, the subscripts of the different blocks yield the matrix partition among the processors, consistently with the node partition shown in Fig. 1: The blocks with the subscript k would be local to the processor k. Only the block H is distributed to all processors. The matrix is thus completely partitioned except for the small block H. For a strict 1D partition (such as the domain in Fig. 1 without the sub-domains 7 and 8), the matrix partition would be complete.

2.2. *The asymmetric block factorization*

The solution of the linear system of equations, Eq. (1), can be obtained in the following way: First, the matrix is factored and then the solution is found by a forward and backward substitution. This two-step procedure is most appropriate for problems with constant coefficient matrices.

The factorization consists in applying successively a LU decomposition to the original matrix as follows

$$\begin{pmatrix} \mathcal{A} & \mathcal{V} & \mathcal{W} \\ \mathcal{V}^T & \mathcal{D} & \mathcal{Z} \\ \mathcal{W}^T & \mathcal{Z}^T & H \end{pmatrix} = \begin{pmatrix} \mathcal{A} & 0 & 0 \\ \mathcal{V}^T & \overline{\mathcal{D}} & \overline{\mathcal{Z}} \\ \mathcal{W}^T & \overline{\mathcal{Z}}^T & \overline{H} \end{pmatrix} \begin{pmatrix} I & \mathcal{A}^{-1}\mathcal{V} & \mathcal{A}^{-1}\mathcal{W} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix},$$

$$\begin{pmatrix} \mathcal{\overline{D}} & \overline{\mathcal{Z}} \\ \overline{\mathcal{Z}}^T & \overline{H} \end{pmatrix} = \begin{pmatrix} \overline{\mathcal{D}} & 0 \\ \overline{\mathcal{Z}}^T & \overline{\overline{H}} \end{pmatrix} \begin{pmatrix} I & \overline{\mathcal{D}}^{-1}\overline{\mathcal{Z}} \\ 0 & I \end{pmatrix},$$

and therefore

$$\begin{pmatrix} \mathcal{A} & \mathcal{V} & \mathcal{W} \\ \mathcal{V}^T & \mathcal{D} & \mathcal{Z} \\ \mathcal{W}^T & \mathcal{Z}^T & H \end{pmatrix} = \begin{pmatrix} \mathcal{A} & 0 & 0 \\ \mathcal{V}^T & \overline{\mathcal{D}} & 0 \\ \mathcal{W}^T & \overline{\mathcal{Z}}^T & \overline{\overline{H}} \end{pmatrix} \begin{pmatrix} I & \mathcal{A}^{-1}\mathcal{V} & \mathcal{A}^{-1}\mathcal{W} \\ 0 & I & \overline{\mathcal{D}}^{-1}\overline{\mathcal{Z}} \\ 0 & 0 & I \end{pmatrix}, \tag{2}$$

where I is the identity matrix and $\overline{\mathcal{D}}$, $\overline{\mathcal{Z}}$, $\overline{H}$ and $\overline{\overline{H}}$ are the Schur complements defined by

$$\begin{aligned} \overline{\mathcal{D}} &= \mathcal{D} - \mathcal{V}^T\mathcal{A}^{-1}\mathcal{V} \\ \overline{\mathcal{Z}} &= \mathcal{Z} - \mathcal{V}^T\mathcal{A}^{-1}\mathcal{W} \\ \overline{H} &= H - \mathcal{W}^T\mathcal{A}^{-1}\mathcal{W} \\ \overline{\overline{H}} &= \overline{H} - \overline{\mathcal{Z}}^T\overline{\mathcal{D}}^{-1}\overline{\mathcal{Z}} \end{aligned} \tag{3}$$

Notice that the computation of the Schur complements, Eq. (3), will introduce *fill-in* in the block diagonal matrices $\mathcal{D}$ and H, as well as the sparse matrix $\mathcal{Z}$. Referring to Fig. 2 and the first equation of (3), the matrix $\overline{\mathcal{D}}$ will have a block tridiagonal structure:

$$\overline{\mathcal{D}} = \begin{pmatrix} \overline{D}_1 & E_1 & & \\ E_1^T & \overline{D}_2 & E_2 & \\ & \ddots & \ddots & \ddots \end{pmatrix}, \tag{4}$$

where the blocks $\overline{D}_k$ and E_k are computed from

$$\begin{aligned} \overline{D}_k &= D_k - V_k^T A_k^{-1} V_k - U_{k+1}^T A_{k+1}^{-1} U_{k+1}, \\ E_k &= \ - U_{k+1}^T A_{k+1}^{-1} V_{k+1}. \end{aligned} \tag{5}$$

Likewise, the Schur complements $\overline{\mathcal{Z}}$ and $\overline{H}$ are obtained by computing the blocks $\overline{Z}_k$ from

$$\overline{Z}_k = Z_k - V_k^T A_k^{-1} W_k - U_{k+1}^T A_{k+1}^{-1} W_{k+1}, \tag{6}$$

and

$$\overline{H} = H - \sum_k W_k^T A_k^{-1} W_k. \tag{7}$$

In all of these operations, the multiplications with the inverse of the matrix blocks A_k are replaced by two solution steps using the Cholesky factorization, $A_k = L_k L_k^T$, since we assumed that A_k is symmetric, positive definite and banded. The matrix multiplications with V_k^T, U_k^T and W_k^T are performed as sparse matrix operations. Notice that the solution steps together with the multiplications can be done locally within the processor k and then sent down to processor $k-1$ to compute the Schur complements $\overline{D}_k$, E_k and $\overline{Z}_k$. The computation of $\overline{H}$ demands however a global sum across the processors.

The Schur complement $\overline{\overline{H}}$ defined in the last equation of (3) can be written explicitely as

$$\overline{\overline{H}} = \overline{H} - \sum_k \overline{Z}_k^t \overline{Z}_k', \tag{8}$$

where $\overline{Z}_k'$ are the solutions of the following block tridiagonal system of equations

$$\begin{pmatrix} \overline{D}_1 & E_1 & & & \\ E_1^T & \overline{D}_2 & E_2 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{n-2}^T & \overline{D}_{n-1} & E_{n-1} \\ & & & E_{n-1}^T & \overline{D}_n \end{pmatrix} \begin{pmatrix} \overline{Z}_1' \\ \overline{Z}_2' \\ \vdots \\ \overline{Z}_{n-1}' \\ \overline{Z}_n' \end{pmatrix} = \begin{pmatrix} \overline{Z}_1 \\ \overline{Z}_2 \\ \vdots \\ \overline{Z}_{n-1} \\ \overline{Z}_n \end{pmatrix}. \tag{9}$$

A parallel scheme to solve this linear system will be described in section 2.4.

It should be noted that the factorization used in Eq. (2), also known as an *asymmetric block factorization* [2], can be easily generalized to an asymmetric and non-positive matrix.

2.3. *The forward and backward substitutions*

The solution of the factored system, Eq. (2), can be readily computed by performing a *forward* substitution

$$\begin{pmatrix} \mathcal{A} & 0 & 0 \\ \mathcal{V}^T & \overline{\mathcal{D}} & 0 \\ \mathcal{W}^T & \overline{\mathcal{Z}}^T & \overline{\overline{H}} \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} b \\ c \\ d \end{pmatrix}, \tag{10}$$

followed by a *backward* substitution

$$\begin{pmatrix} I & \mathcal{A}^{-1}\mathcal{V} & \mathcal{A}^{-1}\mathcal{W} \\ 0 & I & \overline{\mathcal{D}}^{-1}\overline{\mathcal{Z}} \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}. \tag{11}$$

More explicitely, the solutions x, y and z are obtained by successively solving the following systems of equations:

$$\begin{aligned} \mathcal{A}x' &= b, \\ \overline{\mathcal{D}}y' &= c - \mathcal{V}^T x', \\ \overline{\overline{H}}z &= d - \mathcal{W}^T x' - \overline{\mathcal{Z}}^T y', \end{aligned} \tag{12}$$

during the forward step and

$$\begin{aligned} \overline{\mathcal{D}}y &= \overline{\mathcal{D}}y' - \overline{\mathcal{Z}}z, \\ \mathcal{A}x &= \mathcal{A}x' - \mathcal{V}y - \mathcal{W}z, \end{aligned} \tag{13}$$

during the backward step.

The first equation in (12) and the second equation in (13) can be readily solved in parallel since $\mathcal{A}$ is block diagonal. Since the full matrix $\overline{\overline{H}}$ is distributed to all processors, the third equation in (12) is solved in every processor. The solutions of the equations involving the block tridiagonal matrix $\overline{\mathcal{D}}$ are obtained by the parallel cyclic reduction method which is described in the next section.

2.4. *The parallel cyclic reduction*

The block tridiagonal linear system in both the factorization step, Eq. (9) and the solution step, Eqs. (12), (13) has the form

$$\begin{pmatrix} D_1 & E_1 & & & \\ E_1^T & D_2 & E_2 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{n-2}^T & D_{n-1} & E_{n-1} \\ & & & E_{n-1}^T & D_n \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-1} \\ s_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix}, \tag{14}$$

where the bars have been omitted from D_k and s, r denote the solution and the right-hand-side arrays, respectively. Such a system can be solved in many ways. The best-known is the recursive block Gaussian elimination which leads to the system,

$$\begin{pmatrix} \hat{D}_1 & E_1 & & & \\ 0 & \hat{D}_2 & E_2 & & \\ & \ddots & \ddots & \ddots & \\ & & 0 & \hat{D}_{n-1} & E_{n-1} \\ & & & 0 & \hat{D}_n \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-1} \\ s_n \end{pmatrix} = \begin{pmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \vdots \\ \hat{r}_{n-1} \\ \hat{r}_n \end{pmatrix}, \tag{15}$$

where

$$\begin{aligned}\hat{D}_{k+1} &= D_{k+1} - E_k^T \hat{D}_k^{-1} E_k,\\ \hat{r}_{k+1} &= r_{k+1} - E_k^T \hat{D}_k^{-1} \hat{r}_k, \qquad k = 1, \ldots, n-1,\end{aligned} \tag{16}$$

and $\hat{D}_1 = D_1$, $\hat{r}_1 = r_1$. The system Eq. (15) can be solved by simple back-substitution. Both, the elimination and back-substitution procedures are however, entirely recursive and cannot be parallelized.

A similar Gaussian elimination is known under the name of Burn At Both Ends (BABE) algorithm. Here, the elimination proceeds from both ends towards the center where a purely diagonal block can be produced and from where the subsequent back-substitution can start. In this procedure, only two processors can work in parallel; it can however, be optimal even up to 8 processors because the number of operations is small.

An appropriate procedure for higher numbers of processors is cyclic reduction [5]. The basic idea here is to eliminate first all odd blocks ($k = 1, 3, 5, \ldots$), leading to a system for the even numbered unknowns,

$$\hat{E}_{k-1}^T s_{k-1} + \hat{D}_{k+1} s_{k+1} + \hat{E}_{k+1} s_{k+3} = \hat{r}_{k+1}, \quad k = 1, 3, 5, \ldots \tag{17}$$

where

$$\begin{aligned}\hat{D}_{k+1} &= D_{k+1} - E_k^T D_k^{-1} E_k - E_{k+1} D_{k+2}^{-1} E_{k+1}^T,\\ \hat{E}_{k+1} &= -E_{k+1} D_{k+2}^{-1} E_{k+2},\\ \hat{r}_{k+1} &= r_{k+1} - E_k^T D_k^{-1} r_k - E_{k+1} D_{k+2}^{-1} r_{k+2}.\end{aligned} \tag{18}$$

This procedure reduces the number of blocks from n to $n/2$. A second step can be performed, reducing the number of blocks again by two, and so on. After a total of $\log_2 n$ steps, the elimination is done. At the beginning of the process, half of the processors are active, half are idle. Then, a quarter of the processors work, then an eighth, and so on. At the end, all processors but one are idle. This is a large improvement upon the BABE algorithm but one can do even better.

The cyclic reduction procedure can in fact be performed in parallel [5]. This implies performing the matrix transformations, Eqs. (18), in parallel over all matrix blocks. At the end, not only do we have block number $n/2$ ready to solve, but all the blocks at once. The advantage is in the reduction/back-substitution step for the right hand side. Instead of first reducing the right hand side vector and back-substituting afterwards, it is sufficient to reduce only. The solution s of the block tridiagonal system Eq. (14) can then be immediately obtained.

More explicitly, the Parallel Cyclic Reduction (PCR) algorithm can be described as a recursive computation of new matrix blocks $D_k^{(l)}$, $E_k^{(l)}$ for levels $l = 1, 2, \ldots, L = \lceil \log_2 n \rceil$, (the *ceil* function $\lceil x \rceil$ returns the smallest integer not less than x) using

$$\begin{aligned} D_k^{(l)} &= D_k^{(l-1)} - F_k^{(l)} E_k^{T^{(l-1)}} - G_k^{(l)} E_{k-2^{(l-1)}}^{(l-1)} \\ E_k^{(l)} &= -F_k^{(l)} E_{k+2^{(l-1)}}^{(l-1)} \end{aligned} \tag{19a}$$

where

$$\begin{aligned} F_k^{(l)} &= E_k^{(l-1)} \left[D_{k+2^{(l-1)}}^{(l-1)} \right]^{-1} \\ G_k^{(l)} &= E_{k-2^{(l-1)}}^{T^{(l-1)}} \left[D_{k-2^{(l-1)}}^{(l-1)} \right]^{-1} . \end{aligned} \tag{19b}$$

Blocks $D^{(l)}$ and $E^{(l)}$ having subscripts outside the range $[1, n]$ are set equal to zero and the $l = 0$ blocks refer to the original matrices. Thus, considering a matrix with $n = 7$ blocks, the level $l = 1$ through level $l = 3$ PCR steps are illustrated below

$$\begin{pmatrix} D_1 & E_1 & & & & & \\ E_1^T & D_2 & E_2 & & & & \\ & E_2^T & D_3 & E_3 & & & \\ & & E_3^T & D_4 & E_4 & & \\ & & & E_4^T & D_5 & E_5 & \\ & & & & E_5^T & D_6 & E_6 \\ & & & & & E_6^T & D_7 \end{pmatrix} \overset{l=1}{\Longrightarrow} \begin{pmatrix} D_1^{(1)} & & E_1^{(1)} & & & & \\ & D_2^{(1)} & & E_2^{(1)} & & & \\ E_1^{T^{(1)}} & & D_3^{(1)} & & E_3^{(1)} & & \\ & E_2^{T^{(1)}} & & D_4^{(1)} & & E_4^{(1)} & \\ & & E_3^{T^{(1)}} & & D_5^{(1)} & & E_5^{(1)} \\ & & & E_4^{T^{(1)}} & & D_6^{(1)} & \\ & & & & E_5^{T^{(1)}} & & D_7^{(1)} \end{pmatrix}$$

$$\overset{l=2}{\Longrightarrow} \begin{pmatrix} D_1^{(2)} & & & & E_1^{(2)} & & \\ & D_2^{(2)} & & & & E_2^{(2)} & \\ & & D_3^{(2)} & & & & E_3^{(2)} \\ & & & D_4^{(2)} & & & \\ E_1^{T^{(2)}} & & & & D_5^{(2)} & & \\ & E_2^{T^{(2)}} & & & & D_6^{(2)} & \\ & & E_3^{T^{(2)}} & & & & D_7^{(2)} \end{pmatrix}$$

$$\overset{l=3}{\Longrightarrow} \begin{pmatrix} D_1^{(3)} & & & & & & \\ & D_2^{(3)} & & & & & \\ & & D_3^{(3)} & & & & \\ & & & D_4^{(3)} & & & \\ & & & & D_5^{(3)} & & \\ & & & & & D_6^{(3)} & \\ & & & & & & D_7^{(3)} \end{pmatrix}$$

Note that after the first step the result for the even blocks is identical to Eq. (17), the only difference being the notation which has changed from ($\hat{D}_k$, $\hat{E}_k$) to ($D_k^{(1)}$, $E_k^{(1)}$), respectively. The distance between the outer diagonal blocks to the diagonal blocks doubles again after each further PCR step. Altogether, L steps are needed to move all the outer diagonal blocks out of the matrix, resulting in a block diagonal matrix which can then be inverted in parallel. The next phase implies the cyclic reduction of the right-hand side:

$$r_k^{(l)} = r_k^{(l-1)} - F_k^{(l)} r_{k+2^{(l-1)}}^{(l-1)} - G_k^{(l)} r_{k-2^{(l-1)}}^{(l-1)}, \tag{20}$$

and the matrix-vector product

$$s_k = \left[D_k^{(L)}\right]^{-1} r_k^{(L)} \tag{21}$$

to obtain the solution of Eq. (14).

3. Implementation of the parallel solver algorithm

In this section, the full algorithm for solving the matrix problem stated in Eq. (1) is described. The algorithm is divided in a *factorization* phase and a *solution* phase. Such a two-step procedure is most suitable when considering for example, time-dependent simulations that have constant coefficient matrices. The cost of the factorization, which can be a few orders of magnitude higher than that of one solution step, may easily be amortized as the number of these latter steps increases.

3.1. Factorization phase

Referring to the decomposed matrix shown in Eq. (2), the factorization phase can be stated as follows:

1. Factor $A_k = L_k L_k^T$ (SPBTRF).
2. Compute the Schur complements $\overline{D}_k$, E_k, Z_k, $\overline{H}$, using Eqs. (5–7).
3. Perform parallel cyclic reduction of the matrix $\overline{\mathcal{D}}$: for $l = 1, \ldots, L$
   - 3.1. Invert $\overline{D}_k$ (SPOTRF and SPOTRI)
   - 3.2. Compute the PCR matrices $F_k^{(l)}$, $G_k^{(l)}$, $\overline{D}_k^{(l)}$, $E_k^{(l)}$, using Eqs. (19).

4. Invert $\overline{D}_k^{(L)}$ (SPOTRF and SPOTRI).
5. Compute the Schur complements $\overline{\overline{H}}$, using Eq. (8).
6. Invert $\overline{\overline{H}}$ (SPOTRF and SPOTRI).

We have used the LAPACK [6] routines SPBTRF, SPOTRF and SPOTRI to factor and to invert locally the matrices. All these steps can be performed in parallel. The steps involving the computations of the Schur complements and the PCR matrices, steps 2, 3.2 and 5, require communication across processors and are handled with the PVM message passing library [7] send and receive primitives. The optimized BLAS3 matrix multiplication routine SGEMM is used in these steps. The products involving in Eqs. (5–7) the sparse matrices U_k, V_k and W_k are handled with sparse matrix-vector multiplication routines in order to minimize the operation counts. Notice that the global sums are required to compute the Schur complements $\overline{H}$ and $\overline{\overline{H}}$. This operation is performed using the cascade sum algorithm so that the resulting matrix resides on every processing element. At the end of the factorization phase, the matrices

$$\left\{L_k,\quad U_k,\quad V_k,\quad W_k,\quad F_k^{(l)},\quad G_k^{(l)},\quad \left[\overline{D}_k^{(L)}\right]^{-1},\quad \overline{Z}_k,\quad \overline{\overline{H}}^{-1}\right\} \tag{22}$$

required in the solution phase, are stored locally in the processor k. Note that only the last blocks $\left[\overline{D}_k^{(L)}\right]^{-1}$ are needed for the solution phase.

3.2. *Solution phase*

The solution algorithm is separated in the *forward* substitution and the *backward* substitution as specified by Eqs. (12) and (13) respectively:

1. Forward Solve
   1.1. Solve $(L_k L_k^T)x_k = b_k$ (SPBTRS)
   1.2. Compute and send $U_k^T x_k$ to processor $(k-1)$
   1.3. Receive $U_{k+1}^T x_{k+1}$
   1.4. Compute $c_k = c_k - V_k^T x_k - U_{k+1}^T x_{k+1}$
   1.5. Compute $d = d - \sum_k W_k^T x_k$
   1.6. Solve by PCR the block tridiagonal system $\overline{\mathcal{D}}y = c$
   1.7. Compute $d = d - \sum_k Z_k^T y_k$
   1.8. Compute $z = \overline{\overline{H}}^{-1} d$ (SGEMV)

```
2. Backward Solve
    2.1. Compute $t'_k = \overline{Z}_k z$
    2.2. Solve by PCR the block tridiagonal system $\overline{\mathcal{D}} t = t'$
    2.3. Compute $y_k = y_k - t_k$
    2.4. Send $y_k$ to processor $(k+1)$
    2.5. Receive $y_{k-1}$
    2.6. Compute $t'_k = U_k y_{k-1} + V_k y_k + W_k z$
    2.7. Solve $(L_k L_k^T) t_k = t'_k$   (SPBTRS)
    2.8. Compute $x_k = x_k - t_k$
```

The complete solutions x_k, y_k and z are obtained respectively after steps 2.8, 2.3 and 1.8. For simple 1D decomposed domains (i.e., the domain in Fig. 1 without the sub-domains 7 and 8), the matrix H is empty, thus one can skip the steps 1.7, 1.8, 2.1, 2.2 and 2.3. The most demanding communication is in steps 1.5 and 1.7 which require a global sum across the processing elements. The others steps, are either completely local or need only a send/receive to a neighbor.

Assuming that there are n blocks (partitioned among n processing elements), and thus $L = \lceil \log_2 n \rceil$ levels, the PCR algorithm called by steps 1.6 and 2.2, is derived from Eqs. (20) and (21) as follows:

```
1. For $l = 1, L$ and with $m = 2^{(l-1)}$
    1.1. Send $r_k$ to processor $(k-m)$,                    $k = m+1, n$
    1.2. Send $r_k$ to processor $(k+m)$,                    $k = 1, n-m$
    1.3. Receive $r_{k+m}$ and compute $r_k = r_k - F_k r_{k+m}, k = 1, n-m$
    1.4. Receive $r_{k-m}$ and compute $r_k = r_k - G_k r_{k-m}, k = m+1, n$
2. Compute $s_k = [\overline{D}_k^{(L)}]^{-1} r_k$,      $k = 1, n$
```

where s and r are the solution and right-hand side arrays, and k designates the processor number.

3.3. Storage and operation counts

For a geometry as shown in Fig. 3, the total number of points interior to a sub-domain N_I, the number of vertical connectivity points N_V and the number of horizontal points N_H are respectively:

$$\begin{aligned} N_I &\simeq Pmn \\ N_V &\simeq Pn \\ N_H &= h. \end{aligned} \tag{23}$$

Here, P is the number of processors, taken equal to the number of sub-domains, and m, n and h are defined in Fig. 3. The main contribution to the memory space needed for the solver comes from the storage of the different matrices listed in (22). Neglecting the sparse matrices U_k, V_k and W_k, the total memory required *per processor* is

$$M = n\left(m^2 + n + 2n\log_2 P + h + h^2/n\right), \tag{24}$$

where we have assumed that the factored matrices $A_k = L_k L_k^T$ are stored as symmetric band matrices having a half bandwidth of m while the other matrices are considered as full. We also assumed that each processor has a copy of the matrix $\overline{\overline{H}}$ and therefore, memory space could be a problem when N_H is large.

The number of floating-point operations *per processor* for the factorization, T_f and solution phases, T_s, described in sections 3.1 and 3.2 can be estimated as

$$T_f = n\left(m^3 + 11n^2\log_2 P + 4nh\log_2 P + h^3/n\right), \tag{25}$$

$$T_s = n\left(8m^2 + 8n\log_2 P + 4h + 2h^2/n\right). \tag{26}$$

Only the dominant operations are counted here. All of the sparse matrix-vector multiplications are omitted. For a scaled size problem (n, m and h fixed with P increasing), the operation count as well as the storage per processor increase only as $\log_2 P$, while the problem size increases linearly with P. The algorithm can be therefore considered as nearly scalable, as long as the number of horizontal connectivity points N_H does not increase with P or is negligeable compared to N_I and N_V. Otherwise, the parallelization of the dense matrix $\overline{\overline{H}}$ must be considered, e.g. using SCALAPACK [1] routines.

Let us consider a fixed size problem, keeping n, h and $m' = Pm$ fixed. The memory requirement and the operation count are now given by

$$\begin{aligned} M(P) &= M_0 + n\left(m'^2/P^2 + 2n\log_2 P\right), \\ T_f(P) &= T_{f0} + n\left(m'^3/P^3 + 11n^2\log_2 P + 4nh\log_2 P\right), \\ T_s(P) &= T_{s0} + n\left(8m'^2/P^2 + 8n\log_2 P\right). \end{aligned} \tag{27}$$

The number of operations starts to decrease strongly as $1/P^3$ and $1/P^2$ respectively for the factorization and the solution steps. This behavior is due to the chosen node ordering scheme which halves the bandwidth m of the band matrix A_k each time the domain is dissected by two. For a large number of processors, the calculations are dominated by the parallel cyclic reduction which yields a $\log_2 P$ dependency. An estimate of the number of processors which minimizes the operation counts during the solution stage T_s is

$$P_{\text{op}} \sim \left(\frac{2\ln 2}{n}\right)^{1/2} m'. \tag{28}$$

4. Performance measurement

The parallel solver described above is implemented on a Cray-T3D, using the `PVM` [7] send and receive primitives for the inter-processor communications. The single processor optimization is achieved by an systematic use of `LAPACK` and `BLAS` routines. Since these libraries exist in most computers, the portability of the solver is ensured, without a loss of optimization.

The timing of the parallel code is done by measuring the times spent by the factorization and the solution parts on each processor, and then finding their *maximum* across the processors. Note that the times defined in this way include all the communication overhead. The *total* number of operations summed over all processors, is obtained by introducing a counter in the program, yielding thus the computation rate. The matrix considered arises from the discretization of the Poisson equation, using bilinear finite elements.

The performance results for a scaled size problem, with $n = 46$, $m = 8$ and $h = 3m$, (the sizes n, m, h were defined in Fig. 3), are listed in Table 1, together with the total number of unknowns, for a number of processors P varying from 8 up to 256. In agreement with Eqs. (25) and (26), the factorization times and the backsolve times are found to vary as $\log_2 P$, as can be seen in Fig. 4. The flop rates are higher for the factorization stage than for the solution stage, due to the frequent calls to BLAS3 matrix-matrix product routine in the former one. A computation rate of 5.9 GFlops is obtained for the factorization step at 256 processors. We have observed that these rates increase slightly with increasing n and m.

The performance for the fixed size problem, with $m' = 576, n = 46, h = 192$ (see Fig. 3), is shown in Table 2. In this problem, P_1 processors are assigned to the lower $m' \times n$ rectangular domain and $P_2 = P - P_1$ processors are assigned to the upper $h \times n$ domain. There are altogether 35997 unknowns in this problem. Both the factorization and the backsolve times decrease strongly at small values of P, since the number of operations for solving the band linear system decreases, as can be expected from Eqs. (27). The $1/P^2$ decrease of the backsolve time at small P is illustrated in Fig. 5. The estimated optimum value of P calculated from Eq. (28) for this problem is 100 which is somewhat smaller than the value of P_1 observed here. Notice that at $P = 256$, the mesh size of each sub-domain m is only 3.

5. Conclusion

In this paper, we have presented a parallel direct solver for linear systems of the type arising from finite element or difference discretizations of partial differential equations in two dimensions. Using an adequate ordering scheme for the discretized unknowns, the pertinent matrix can always be brought into the form shown in Fig. 2. Throughout the paper, the matrix has been assumed to be symmetric and positive definite; this restriction is however, not necessary for the application of the present method. It can easily be extended to general, asymmetric matrices.

The method utilizes the domain decomposition technique for the parallelization and is shown to be efficient when the decomposition is mostly performed in one direction. For very thin and long geometries, the method scales as $\log_2$ of the number of processors P at large values of P. Moreover, the solver is separated in a factorization stage and a solution stage, making it very attractive for problems where the same linear system is solved repeatedly for different right-hand sides as e.g. in time-dependent simulations. In fact, our main motivation came from 2D Particle-In-Cell (PIC) electrostatic simulations where Poisson's equation must be solved at every time step [8–9].

The implementation uses the LAPACK/BLAS [6] library to optimize matrix operations on single processors and the PVM library [7] for communication to produce an optimized and portable code. Test runs on the 256 processor Cray T3D system show that it is possible to solve a Poisson equation on a 2D grid of 96000 nodes in about 12 ms per backsolve step. A computation rate of 5.9 GFlops during the factorization stage is achieved at 256 processors.

Acknowledgments

This work was supported in part by the Swiss National Science Foundation and by Cray Research, Inc. within the framework of the Cray Research/EPFL Parallel Application Technology Program. The computations were done on the Cray T3D massively parallel computer at EPFL, Lausanne. The authors would like to acknowledge O. Sauter and S. Merazzi for fruitful discussions and D. Whaley for reading the manuscript.

References

[1] J. Choi, J.J. Dongarra, D. Walker, R.C. Whaley, " SCALAPACK Reference Manual", Version 1.0 Beta, ORNL/TM-12470, 1994.

[2] A. George, J.W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall Inc., 1981.

[3] G. Strang, G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewoods Cliffs, 1973.

[4] T.M. Tran, K. Appert, O. Sauter, "A Direct Poisson Solver for Particle-in-Cell (PIC) Simulation", CRPP-EPFL/LRP 499/94, 1994.

[5] R.W. Hockney, C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd., 1985.

[6] E . Anderson *et al.*, *LAPACK User's Guide*, SIAM, 1992.

[7] G.A. Geist, A.L. Beguelin, J.J. Dongarra, R.J. Mancheck, V.S. Sunderam, "PVM 3.0 User's Guide and Reference Manual", ORNL/TM-12187, 1993.

[8] C.K. Birdsall, A.B. Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill Inc., 1985.

[9] R.W. Hockney, J.W. Eastwood, *Computer Simulation using Particles*, Adam Hilger Inc., 1988.

Table 1: Performance of the parallel matrix solver on the Cray-T3D for a scaled size problem, with $n = 46, m = 8, h = 3m$.

P	Number of unknowns	FACT (s)	FACT (MFlops)	SOLVE (ms)	SOLVE (MFlops)
8	3077	0.228	74.9	6.16	66.0
16	6085	0.296	221.9	7.69	157.3
32	12101	0.344	554.5	9.03	352.1
64	24133	0.385	1270.5	10.09	768.8
128	48197	0.423	2788.3	11.08	1634.0
256	96325	0.463	5926.4	12.11	3400.1

Table 2: Performance of the parallel matrix solver on the Cray-T3D for a fixed size problem, with $m' = 576, n = 46, h = 192$.

P	P_1	$P_2 =$ $P - P_1$	FACT (s)	FACT (MFlops)	SOLVE (ms)	SOLVE (MFlops)
4	3	1	128.75	40.5	1109.5	49.8
8	6	2	25.32	80.4	284.5	99.9
16	12	4	5.40	197.4	81.1	198.5
32	24	8	2.17	457.4	29.8	411.9
64	48	16	1.61	1018.1	16.6	938.7
128	96	32	1.60	2122.1	13.4	2125.2
256	192	64	1.74	4254.3	13.2	4449.4

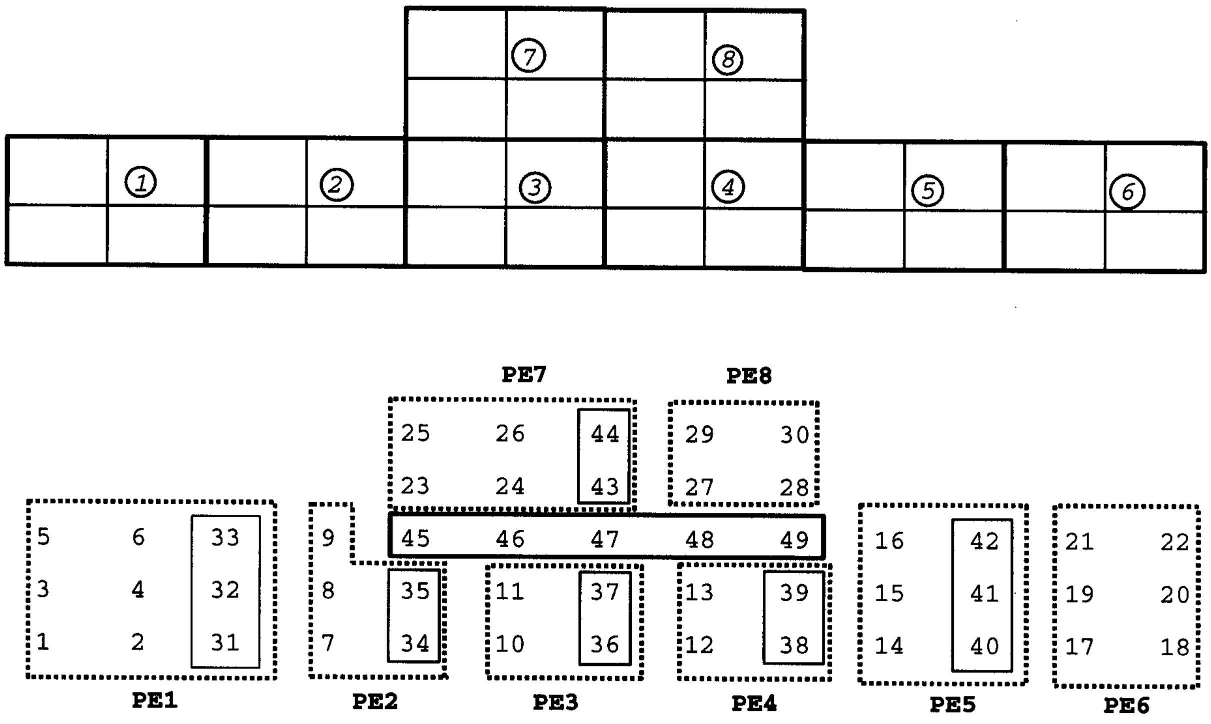


Fig. 1: A 2×2 cell (per sub-domain) mesh and the node numbering for a 8 sub-domain case. The node partition among the processors is represented by boxes with dashed borders.

$$
\left[\begin{array}{cccccccc|ccccccc|c}
A_1 & & & & & & & & V_1 & & & & & & & W_1 \\
 & A_2 & & & & & & & U_2 & V_2 & & & & & & W_2 \\
 & & A_3 & & & & & & & U_3 & V_3 & & & & & W_3 \\
 & & & A_4 & & & & & & & U_4 & V_4 & & & & W_4 \\
 & & & & A_5 & & & & & & & U_5 & V_5 & & & W_5 \\
 & & & & & A_6 & & & & & & & U_6 & [V_6] & & W_6 \\
 & & & & & & A_7 & & & & & & & [U_7] & V_7 & W_7 \\
 & & & & & & & A_8 & & & & & & & U_8 & W_8 \\
\hline
V_1^T & U_2^T & & & & & & & D_1 & & & & & & & Z_1 \\
 & V_2^T & U_3^T & & & & & & & D_2 & & & & & & Z_2 \\
 & & V_3^T & U_4^T & & & & & & & D_3 & & & & & Z_3 \\
 & & & V_4^T & U_5^T & & & & & & & D_4 & & & & Z_4 \\
 & & & & V_5^T & U_6^T & & & & & & & D_5 & & & Z_5 \\
 & & & & & [V_6^T] & [U_7^T] & & & & & & & [D_6] & & [Z_6] \\
 & & & & & & V_7^T & U_8^T & & & & & & & D_7 & Z_7 \\
\hline
W_1^T & W_2^T & W_3^T & W_4^T & W_5^T & W_6^T & W_7^T & W_8^T & Z_1^T & Z_2^T & Z_3^T & Z_4^T & Z_5^T & [Z_6^T] & Z_7^T & H
\end{array}\right]
$$

Fig. 2: The matrix structure for the example of Fig. 1. Blocks within brackets are empty.

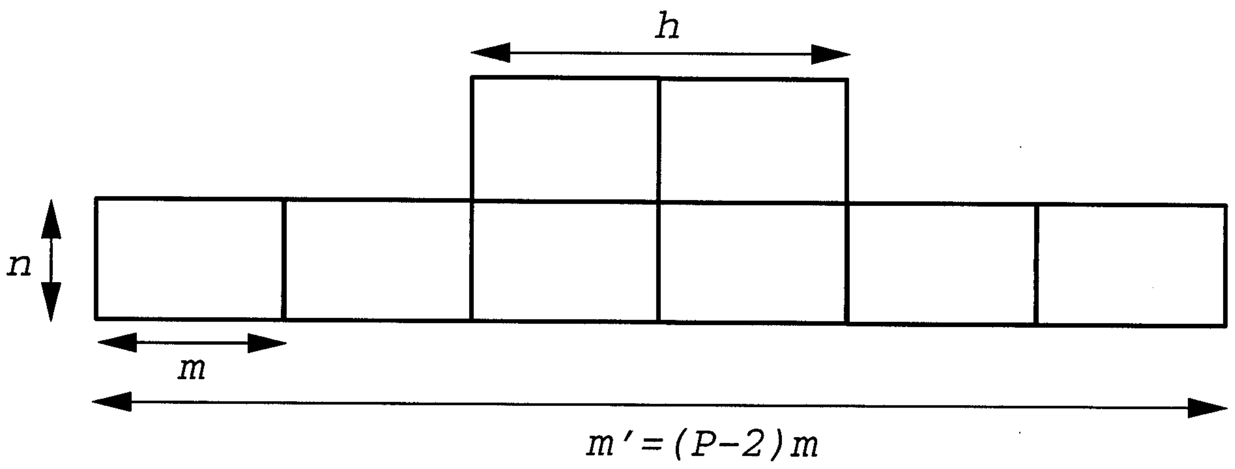
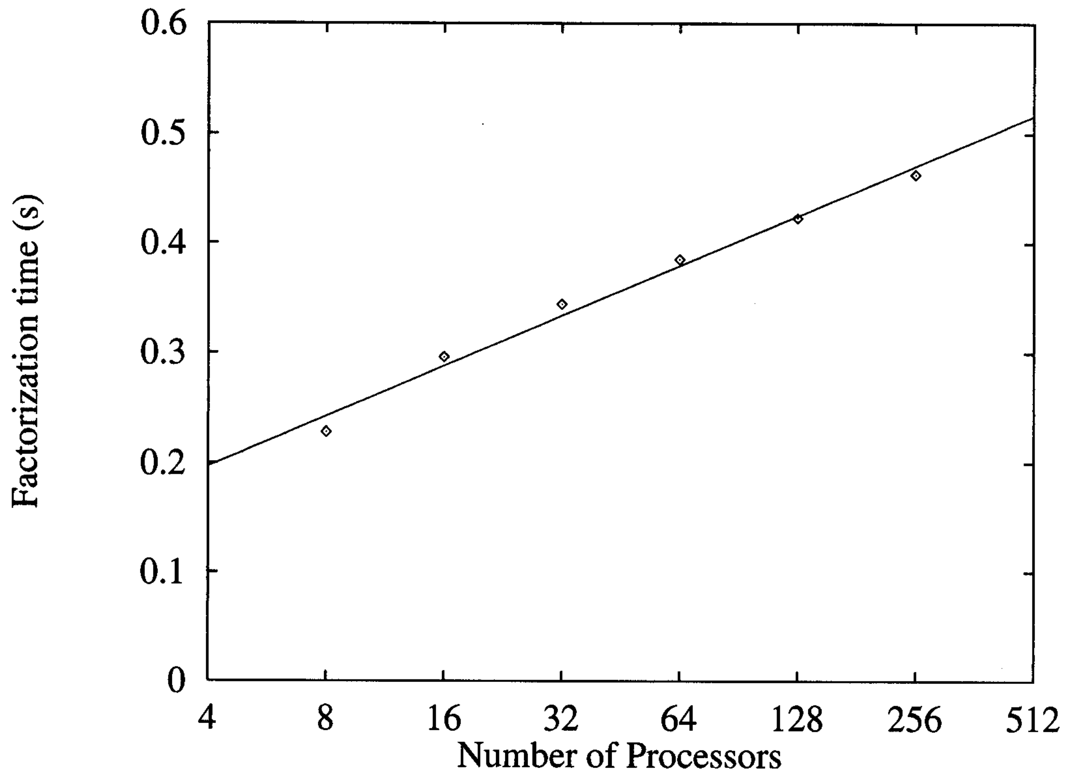
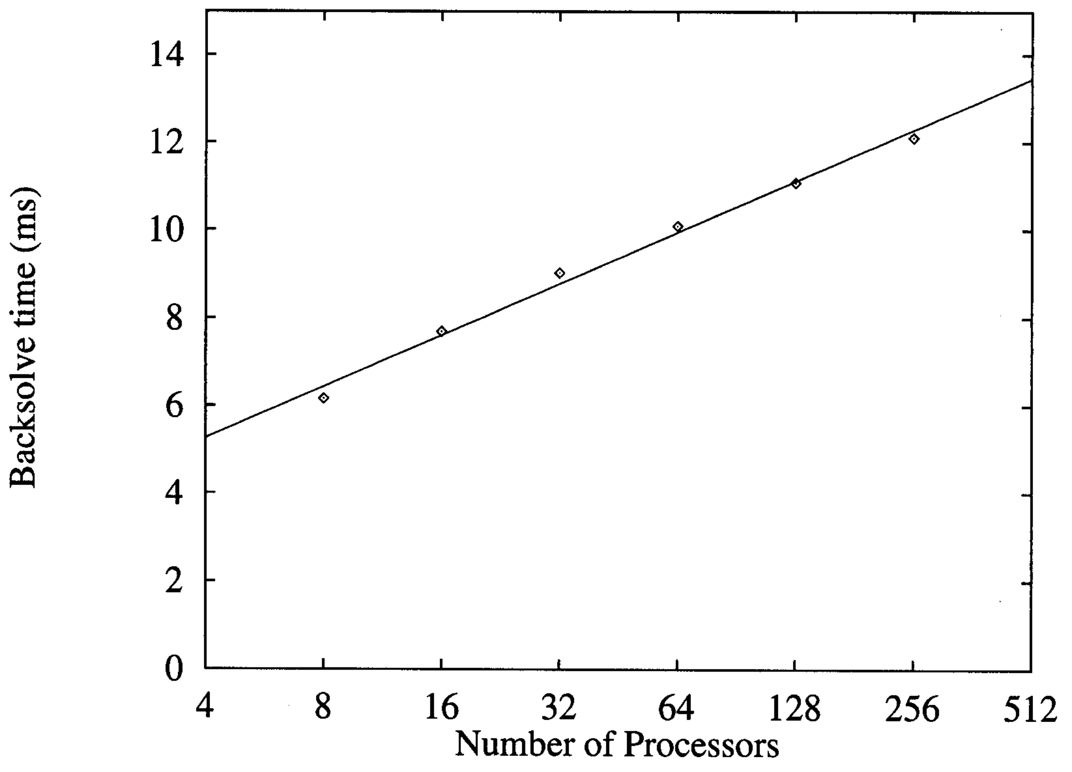


Fig. 3: A geometrical domain considered in the operation counts: each sub-domain has $m \times n$ nodes, h is the number of "horizontal" connectivity points and P designates the number of processing elements, which is the same as the number of sub-domains.



(a)



(b)

Fig. 4: Computational time spent during the factorization stage (a) and the backsolve stage (b) for a scaled size problem with with $n = 46, m = 8, h = 3m$.

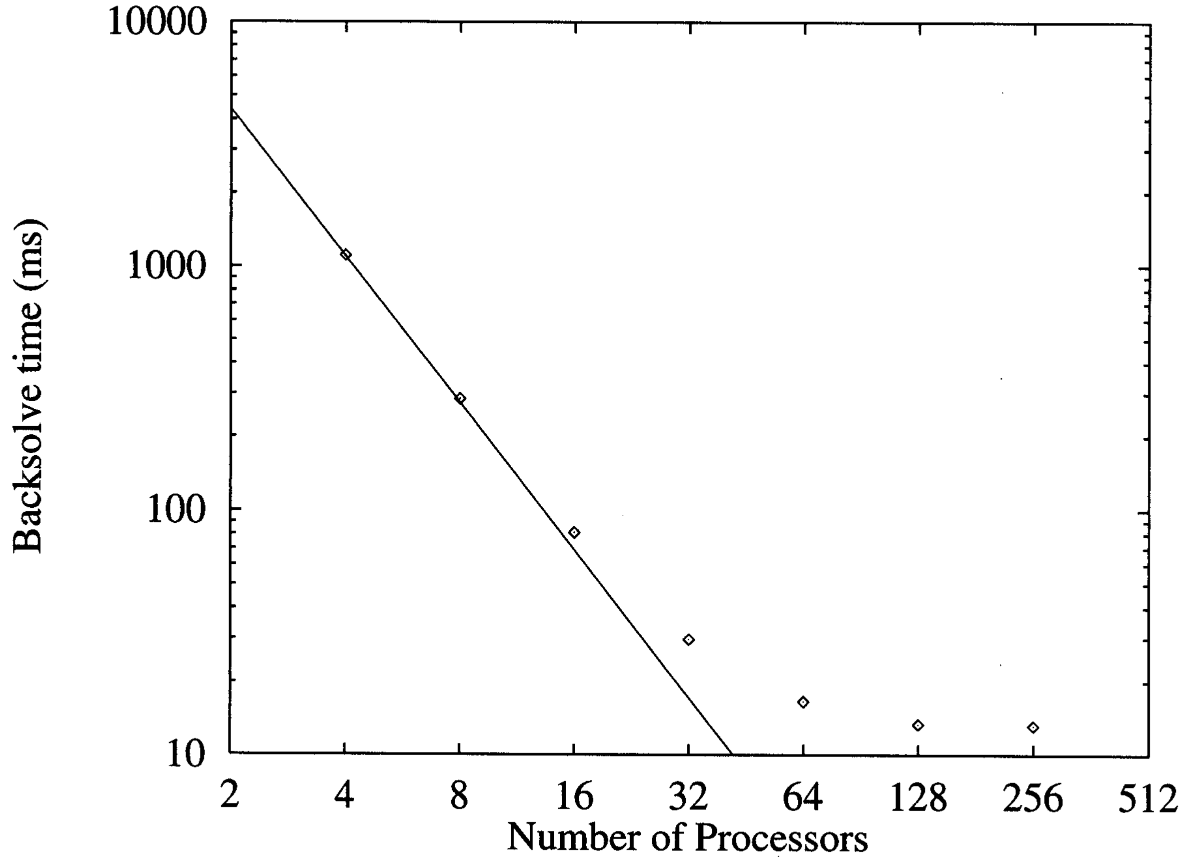


Fig. 5: Computation time spent during the backsolve stage for a fixed size problem with $m' = 576, n = 46, h = 192$. The solid line represents the $1/P^2$ dependency