A Parallel Spline Collocation-Capacitance Method for Elliptic Parallel Differential Equations

C. C. Christara

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
88-735
A PARALLEL SPLINE COLLOCATION-CAPACITANCE METHOD FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

C.C. Christara
E.N. Houstis
J. R. Rice

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR-735
January 1988
A PARALLEL SPLINE COLLOCATION - CAPACITANCE METHOD
FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

C.C. Christara, E.N. Houstis and J.R. Rice
Purdue University
Department of Computer Science
West Lafayette, IN 47907

Technical Report CSD-TR-735
CAPO Report CER-88-14
January 1988

ABSTRACT

We consider the integration of a domain decomposition technique with a new quadratic spline collocation discretization scheme for solving second order elliptic boundary value problems on rectangles. The domain decomposition method is based on the capacitance matrix technique. Due to the limitations of existing methods for solving the corresponding capacitance problem, we develop and analyze iterative methods for its solution. The optimum partitioning and mapping of the underlying computation is studied on hypercube architectures. A numerical realization of this method is presented on NCUBE/7 (128 processors) and its comparative efficiency is measured. The resulting parallel quadratic spline collocation-capacitance method is seen to be efficient in achieving accurate solutions and in using parallel architectures.

This research supported in part by NSF grants CCR-8704826, AFOSR grant 84-0385, ARO grant DAAL03-86-K-0106, David Ross Foundation and ESPRIT project 1588.
1. INTRODUCTION

In this paper we study a domain decomposition method with a quadratic spline collocation discretization method [5] for solving the second order partial differential equation (PDE)

\[ Lu = aD_x^2 u + bD_x D_y u + cD_y^2 u + dD_x u + eD_y u + fu = g \quad \text{in} \quad \Omega \]

subject to mixed type boundary conditions

\[ Bu = \alpha u + \beta D_u u = g_0 \quad \text{on} \quad \partial \Omega \]

where \( \Omega = [ax, bx] \times [ay, by] \) is a rectangular domain, \( a, b, c, d, e, f, \alpha, \beta, g, g_0 \) are functions of \( x \) and \( y \) in \( C^1(\Omega) \), \( D_u u \) is the normal derivative of \( u \) and \( \partial \Omega \) is the boundary of \( \Omega \).

The chosen discretization schemes have been shown to be very effective for this class of boundary value problems [5]. In Section 2, we briefly describe them and present some data which demonstrate their efficiency. So far there is limited information about the effectiveness of parallel methods for such discretized equations. In Section 3 we define the parallel quadratic spline collocation-capacitance method. We present a domain decomposition method for solving the resulting equations using a capacitance matrix technique [8] because of its inherent parallelism. This technique reduces the work required to solve the so called capacitance system. This is often done by a conjugate gradient (CG) method with appropriate preconditioners. However, in our case we were not able to apply CG successfully. Thus in Section 3.3 we develop and analyze new iterative techniques to solve the capacitance matrix problem.

One of our objectives is to use these methods on MIMD parallel architectures and to determine optimal partitions for the underlying computation. In Sections 4 and 5 we accomplish this for a hypercube machine, the NCUBE/7 with 128 processors. Finally, in Section 6 we present numerical data that illustrates the good efficiency of this method on hypercube architectures.

1. THE QUADRATIC SPLINE COLLOCATION METHOD

Let \( \Delta_x = \{x_k = ax + kh_x; k = 0, \ldots, M\} \), with \( h_x = \frac{bx - ax}{M} \) be a uniform partition of the interval \([ax, bx]\) and \( \Delta_y = \{y_l = ay + lh_y; l = 0, \ldots, N\} \), with \( h_y = \frac{by - ay}{N} \) a uniform partition of \([ay, by]\). Throughout we denote by \( \Delta = \Delta_x \times \Delta_y \) the induced grid partition of \( \Omega \) and by \( \tau_i^j \), \( i = 1, \ldots, M \) the midpoints of \( \Delta_x \) and by \( \tau_j^i \), \( j = 1, \ldots, N \) the midpoints of \( \Delta_y \). For convenience we extend the notation so that \( \tau_0^0 = x_0, \tau_{M+1}^1 = x_M, \tau_0^N = y_0, \tau_{N+1}^1 = y_N \). For later use we define the following sets of points: \( T = \{(\tau_i^j, \tau_j^i), \text{ where } i = 0, \ldots, M, j = 0, \ldots, N\} \) the set of collocation points, \( T_i = \{(\tau_i^j, \tau_j^i), \text{ where } i = 0, \ldots, M, j = 0, \ldots, N\} \) the set of interior collocation points, \( T_{ie} = \{(\tau_i^j, \tau_j^i), \text{ where } i = 0, \ldots, M-1, j = 2, \ldots, N-1\} \subset T \) the set of interior collocation points of \( \Omega \), \( T_b = T \cap \partial \Omega \) the set of boundary collocation points in \( T \) and \( T_{ib} = T \setminus (T_i \cup T_{ie} \cup T_b) \) the subset of interior-boundary collocation points.
Throughout, we denote by $S$ the quadratic spline interpolant of the true solution $u$ of the PDE problem (1.1), defined by the interpolation relations

$$S = u \text{ on } T - T_b,$$

$$S = u - \frac{h_x^4}{128} D_x^4 u \text{ on } T_b \cap \{x = x_0, x_N\},$$

and

$$S = u - \frac{h_y^4}{128} D_y^4 u \text{ on } T_b \cap \{y = y_0, y_N\}.$$

By definition, $S$ belongs to $S_{2,\Delta} = P_{2,\Delta} \cap C^1(\Omega)$, where $P_{2,\Delta}$ is the tensor product of one-dimensional piecewise quadratic polynomials in $x$ and $y$ over the partitions $\Delta_x$ and $\Delta_y$, respectively. $S_{2,\Delta}$ will be referred throughout as the quadratic spline space in two dimensions.

The quadratic spline collocation method introduced in [5] is defined in terms of the following discretization operators. For each interior collocation point in $T_i$, $P_L S$ is defined by the stencil

$$\frac{1}{24} \begin{bmatrix}
  c D_y^2 & S_{i,j+1} \\
 -b D_x D_y & S_{i,j+1} \\
 -e D_y & S_{i,j+1} \\
 -2a D_x^2 & S_{i,j} \\
 a D_x^2 & S_{i-1,j} \\
 -b D_x D_y & S_{i,j} \\
 -2c D_y^2 & S_{i+1,j} \\
 -d D_x & S_{i,j+1} \\
 2d D_x & S_{i,j} \\
 +2e D_y & S_{i,j} \\
 c D_y & S_{i,j-1} \\
 -b D_x D_y & S_{i,j-1} \\
 -e D_y & S_{i,j-1}
\end{bmatrix}.$$
Then $P_{LS}$ is defined by similar stencils at the rest of the interior boundary collocation points in $T_{ib}$ corresponding to $x = \tau_{ib}^x$, $y = \tau_{ib}^y$ and $y = \tau_{ib}^x$. Further, $P_{LS}$ is defined at the corner collocation point $(\tau_{ib}^x, \tau_{ib}^y)$ by the stencil

$$
\begin{bmatrix}
-bD_x D_y S_{1,j+1} \\
+c D^2_y S_{1,j+1} \\
-e D_y S_{1,j+1} \\
+2a D^2_x S_{1,j} \\
-5a D_x^2 S_{2,j} \\
+4a D_x^2 S_{3,j} \\
-a D_x^2 S_{4,j} \\
+5b D_x D_y S_{2,j} \\
-4b D_x D_y S_{3,j} \\
+b D_x D_y S_{4,j} \\
-2c D^2_y S_{1,j} \\
-2d D_x S_{1,j} \\
+5d D_x S_{2,j} \\
-4d D_x S_{3,j} \\
+d D_x S_{4,j} \\
+2e D_y S_{1,j} \\
-b D_x D_y S_{1,j-1} \\
+c D^2_y S_{1,j-1} \\
-e D_y S_{1,j-1}
\end{bmatrix}
$$

$$
\frac{1}{24}
$$

Similar stencils define $P_{LS}$ at the rest of the corner collocation points in $T_{ie}$. Finally, for the
boundary collocation points on the boundary line \( x = ax \), \( P_B S \) is determined by the stencil

\[
\frac{1}{24} \begin{bmatrix}
5\beta D_x S_{1,j} & -13\beta D_x S_{2,j} & +11\beta D_x S_{3,j} & -3\beta D_x S_{4,j}
\end{bmatrix}
\]

Similar stencils define \( P_B S \) in the rest of the boundary collocation points corresponding to the boundary lines \( x = bx, y = ay \) and \( y = by \).

In [5] we proved the following lemma.

Lemma 1. If \( u \) belongs to \( C^6(\Omega) \) then

\[
L_S = g + O(h^2) \quad \text{on} \quad T - T_b,
\]
\[
B_S = g_0 + O(h^2) \quad \text{on} \quad T_b
\]

and

\[
L'S = g + O(h^4) \quad \text{or} \quad L_S = g - P_LS + O(h^4) \quad \text{on} \quad T - T_b,
\]
\[
B'S = g_0 + O(h^4) \quad \text{or} \quad B_S = g_0 - P_B S + O(h^4) \quad \text{on} \quad T_b
\]

where \( L'S = L_S + P_LS, B'S = B_S + P_B S, P_LS \) and \( P_B S \) are perturbation terms defined by the discretization operators above.

2.1 Formulation of the Quadratic Spline Collocation Method

The relations (2.1)–(2.2) lead to three different formulations of the quadratic collocation method. Throughout, they are referred with the acronyms P2C1COL, P2C1CL1 and P2C1CL2.

The first is the standard quadratic spline collocation method. In this case the quadratic spline approximation \( u_\Delta \) to the true solution \( u \) of (1.1) is forced to satisfy

\[
P2C1COL:
\]
\[
Lu_\Delta = g \quad \text{on} \quad T - T_b,
\]
\[
Bu_\Delta = g_0 \quad \text{on} \quad T_b.
\]

This scheme gives second order convergence [5] which follows from Lemma 1.

The fourth order quadratic spline collocation method [5] has \( u \) defined by the extrapolated scheme
A mathematically equivalent but computationally advantageous version of this method is the two step deferred correction method defined by

\begin{align}
\text{P2C1CL2: (1st step)} & \quad L\nu = g \quad \text{on} \quad T - T_b, \\
& \quad B\nu = g_0 \quad \text{on} \quad T_b, \\
\text{(2nd step)} & \quad L\Delta u = g - P_L\nu \quad \text{on} \quad T - T_b, \\
& \quad B\Delta u = g_0 - P_B\nu \quad \text{on} \quad T_b.
\end{align}

Figures 2.1, 2.2 show the structure of the collocation matrices corresponding to equations 2.3 (or 2.5a and 2.4), respectively. Equations (2.3) have at most 9 non-zero elements per row and lower and upper bandwidth \( N + 3 \), while equations (2.5) have at most 27 non-zero elements per row and lower and upper bandwidth \( 5N + 11 \).

Figure 2.1. Structure of the matrix of collocation equation corresponding to P2C1COL for \( N = M = 5 \). \( x \) denotes a non-zero off diagonal element, \( d \) a non-zero diagonal one, while all zero entries are represented by "••" character.
2.2 Sequential Solution of the Quadratic Spline Collocation Equations

In this section we present some numerical results indicating the computational efficiency of various linear algebraic equation solvers for the equations (2.4) and (2.5a). All computations in this section were carried out in double precision on a VAX 8600. A complete scientifically based experimental study of their performance is presented in [9]. Table 2.1 indicates typical performance of several direct and iterative methods for a general elliptic PDE. It is interesting to observe that the iterative methods are applicable to such classes of equations and they become very competitive both in memory and processing time for large grids.

Table 2.2 compares the performance of these spline collocation methods and some Galerkin methods for the problem

\[
Lu = (e^{xy} u_x)_x + (e^{-xy} u_y)_y - \frac{u}{1 + x + y} \quad \text{in} \quad \Omega = [0,1] \times [0,1],
\]

\[
Bu = u \quad \text{on} \quad \partial \Omega
\]

whose true solution is \(u = 0.75e^{xy} \sin(x) \sin(y)\). The collocation equations of P2C1CL1 are solved with Envelope LDU, the ones of P2C1CL2 are solved with Band GE No Piv, while for the Galerkin ones we have applied Envelope LDLT.
Table 2.1. Time in seconds to solve the collocation equations (2.4) and (2.5a) using the indicated direct and iterative methods. The equations were obtained by applying P2C1COL and P2C1CL1 on a general elliptic PDE with a 29x29 uniform grid.

<table>
<thead>
<tr>
<th>Method Solver*</th>
<th>P2C1CL1</th>
<th>P2C1CL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSOR SI</td>
<td>14.3</td>
<td>5.4</td>
</tr>
<tr>
<td>Envelope LDU</td>
<td>22.0</td>
<td>4.3</td>
</tr>
<tr>
<td>Band GE No Piv</td>
<td>54.3</td>
<td>4.9</td>
</tr>
<tr>
<td>Sparse GE No Piv</td>
<td>33.8</td>
<td>6.2</td>
</tr>
<tr>
<td>Linpack Band</td>
<td>75.0</td>
<td>5.3</td>
</tr>
</tbody>
</table>

Table 2.2. Times in seconds for the solution of problem (2.6) using the indicated finite element methods and grids.

<table>
<thead>
<tr>
<th>Method</th>
<th>P2C1CL1</th>
<th>P2C1CL2</th>
<th>Galerkin(2,1)</th>
<th>Galerkin(3,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5x5</td>
<td>0.167</td>
<td>0.105</td>
<td>0.147</td>
<td>0.351</td>
</tr>
<tr>
<td>9x9</td>
<td>0.662</td>
<td>0.324</td>
<td>0.564</td>
<td>1.366</td>
</tr>
<tr>
<td>17x17</td>
<td>4.034</td>
<td>1.660</td>
<td>2.913</td>
<td>7.332</td>
</tr>
<tr>
<td>33x33</td>
<td>39.370</td>
<td>13.715</td>
<td>22.602</td>
<td>54.683</td>
</tr>
</tbody>
</table>

The main objective of this paper is to present and study a class of domain decomposition methods for the solution of spline collocation equations using a capacitance matrix technique or Schur complement method. These are attractive because their inherent parallelism allows us to have efficient parallel implementation on MIMD architectures. For completeness, for four methods we include Table 2.3 which show the errors on the grid points when solving problem (2.6) and the respective orders of convergence. The results of Tables 2.1–2.3 are in agreement with the theoretical analysis of the methods [5] and indicate that spline collocation methods are efficient alternatives for solving general second order elliptic PDEs.

* Abbreviation of methods (see [13] for more details):
  SSOR SI: SOR iteration accelerated by semi-iteration.
  Envelope LDU: An LDU factorization for matrices in envelope form.
  Band GE No Piv: Modified version of Linpack Band.
  Linpack Band: An LU factorization with partial pivoting for banded matrices.
  Galerkin (k,l): Galerkin method for self-adjoint problems based on k degree splines with l continuity.
Table 2.3. Errors and order of convergence for problem (2.6) using the quadratic spline collocation methods and the corresponding Galerkin method.

<table>
<thead>
<tr>
<th>Method</th>
<th>P2C1COL</th>
<th>P2C1CL1</th>
<th>P2C1CL2</th>
<th>Galerkin (2.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5x5</td>
<td>2.4-2</td>
<td>2.7-3</td>
<td>2.8-3</td>
<td>5.9-3</td>
</tr>
<tr>
<td></td>
<td>1.98</td>
<td>3.32</td>
<td>3.32</td>
<td>4.25</td>
</tr>
<tr>
<td>9x9</td>
<td>6.1-3</td>
<td>2.7-4</td>
<td>2.8-4</td>
<td>3.1-4</td>
</tr>
<tr>
<td></td>
<td>2.02</td>
<td>3.91</td>
<td>3.81</td>
<td>4.11</td>
</tr>
<tr>
<td>17x17</td>
<td>1.5-3</td>
<td>1.8-5</td>
<td>2.0-5</td>
<td>1.8-5</td>
</tr>
<tr>
<td></td>
<td>2.10</td>
<td>3.91</td>
<td>3.94</td>
<td>4.03</td>
</tr>
<tr>
<td>33x33</td>
<td>3.5-4</td>
<td>1.2-6</td>
<td>1.3-6</td>
<td>1.1-6</td>
</tr>
</tbody>
</table>

3. A PARALLEL QUADRATIC SPLINE COLLOCATION - CAPACITANCE METHOD

First, we present briefly the idea of the capacitance matrix method for a general system $Ax = b$ with $K$ equations. This method is based on partitioning $Ax = b$ into

$$A_{00} x_0 + A_{01} x_1 = b_0$$

$$A_{10} x_0 + A_{11} x_1 = b_1$$  \hspace{1cm} (3.2a)

$$ (3.2b)$$

where $A_{00}$ is an $n_0 \times n_0$ matrix $x = (x_0, x_1)^T$ $b = (b_0, b_1)^T$. We choose $n_0 < < K$ so that the system $A_{11} x = r$ is easily solvable as compared to $Ax = b$. In the context of solving elliptic PDEs, we decompose $\Omega$ into subdomains and renumber the unknowns and equations so that the unknowns $x_0$ correspond to the boundaries of the subdomains. If the domains contain large numbers of discretization points or elements, then the condition $n_0 < < K$ is satisfied. The simplest decomposition of $\Omega$ that leads to the above partition of the system is that involving two vertical strips, say $\Omega_1, \Omega_2$, where $x_0$ is the vector of unknowns that belong to the middle line that separates $\Omega_1$ from $\Omega_2$ and $x_1$ are the rest of the unknowns. After the elimination of $x_1$ from (3.2b) and its substitution in (3.2a) we obtain the matrix problem

$$C x_0 = (A_{00} - A_{01} A_{11}^{-1} A_{10}) x_0 = b_0 - A_{01} A_{11}^{-1} b_1 = w.$$  \hspace{1cm} (3.3)

The coefficient matrix $C$ is known as the capacitance matrix. After solving (3.3) for $x_0$ one can compute $x_1$ from

$$A_{11} x_1 = b_1 - A_{10} x_0.$$  \hspace{1cm} (3.4)

It is worth noticing that $A_{11}$ is a relatively large well structured matrix, while $C$ is relatively
small, but dense. The computation of $C$ involves the computation of $A_{11}^{-1}$ which is expensive and should be avoided. In the case of positive definite systems, conjugate gradient method (CG) with appropriate preconditioners is usually applied for the solution of (3.3) [10]. In the case of the spline collocation capacitance matrix problem, the CG method does not seem to converge. At least, so far, we are not able to find appropriate preconditioners.

In the rest of the paper, we consider the integration of the capacitance matrix technique with the P2C1COL and P2C1CL2 discretization schemes and its implementation on shared and non-shared memory MIMD machines.

3.1 Domain Decomposition Ordering of the Collocation Equations

In order to apply the capacitance matrix method for the solution of (2.3), we reorder the collocation equations so that the system $A_{11} x_1 = r$ in (3.2) is easily solvable. For the reordering of the (2.3) equations, we assume a decomposition of $\Omega$ in $P = MP \times NP$ rectangular subdomains and number them from bottom up and then from left to right. Throughout we implicitly assume that the computation associated with each subdomain will be allocated to a separate processor.

In the formulation of the collocation equations, the ordering coincides with the ordering of the collocation points. Thus to obtain the decomposition (3.2), it is sufficient to order the collocation points appropriately. We first number the $n_0$ collocation points that lie on subdomain boundaries. Their numbering is irrelevant up to this point. Then we number the rest of the points, i.e., the interior or boundary collocation points of each subdomain, first by the numbering of the subdomains and then numbering the points of each subdomain from left to right and then bottom up.

Figure 3.1a depicts the structure of the matrix of collocation equations with the original ordering (suitable for sequential solution of the system) and Figure 3.1b shows the reordering described above suitable for the capacitance matrix method.

3.2 The Quadratic Spline Collocation-Capacitance Method

With the above reordering of the spline collocation equations, the system is decomposed into four main parts

$$
\begin{bmatrix}
A_{00} & A_{01} \\
A_{10} & A_{11}
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1
\end{bmatrix} =
\begin{bmatrix}
b_0 \\
b_1
\end{bmatrix}
$$

The order of this system is $K = (M+2)(N+2)$ and $A_{00}$ is an $n_0 \times n_0$ sparse matrix with $n_0 = (M+2)(NP - 1) + (N+2)(MP - 1) - (MP - 1)(NP - 1)$, $A_{01}$ is an $n_0 \times n_1$ sparse matrix with $n_1 = K - n_0$, $A_{10}$ is an $n_1 \times n_0$ sparse matrix and $A_{11}$ is an $n_1 \times n_1$ block diagonal matrix with each block being a banded matrix whose bandwidth is $(M+3)/MP$ and having $((N+3)/NP - 1) \times ((M+3)/MP - 2)$ rows. Then the quadratic spline collocation-capacitance method for the collocation equations is defined by the following steps:
Figure 3.1(a). Structure of the matrix of collocation equations (2.3) for $N = M = 5$ grid with the original ordering. $d$, $x$, $\cdot$ are defined as in Figure 2.1.

Figure 3.1(b). Structure of the matrix of collocation equations for $N = M = 5$ grid and a $2 \times 2$ domain decomposition renumbering. $d$, $x$, $\cdot$ are defined as in Figure 2.1.
1. Solve $A_{11} \vec{x}_1 = b_1$.
2. Compute $w = b_0 - A_{01} \vec{x}_1$.
3. Solve $C x_0 = w, C = A_{00} - A_{01} A_{11}^{-1} A_{10}$.
4. Solve $A_{11} x_1 = b_1 - A_{10} x_0$.

For the parallel implementation of this algorithm we observe that the computation in Step 1, i.e. the solution of $A_{11} \vec{x}_1 = b_1$ is equivalent to solving $A_{11}^p \vec{x}_1^p = b_1^p$, for $p = 1, \ldots, P$, where $A_{11}^p$ is the $p$-th block of $A_{11}$ associated with the $p$-th subdomain or processor. Further, the computations of Step 2 and 4 involve the evaluation of the product $A_{01} v_1$ and $A_{10} v_2$ for some vectors $v_1$ and $v_2$. Assuming that each processor knows the vectors $v_1$ and $v_2$, this can be carried out in parallel in a straightforward manner. Thus the efficiency of this method depends very much on the computation of Step 3 and its parallel implementation.

### 3.3 Iterative methods for solving the capacitance matrix system

For the solution of the capacitance system $C x_0 = w$ in Step 3 of the method, we attempted the conjugate gradient method without much success. Thus we introduce a Jacobi-type iterative scheme that avoids the explicit computation of $C$. This scheme is based on the observation that $C x_0 = w$ is equivalent to $A_{00} x_0 = w + A_{01} A_{11}^{-1} A_{10} x_0$. If we denote by $D_{00}$ the diagonal matrix consisting of the diagonal entries of $A_{00}$, then $C x_0 = w$ is equivalently written in the form

$$
D_{00} x_0 = w - C x_0 + D_{00} x_0.
$$

Starting with some initial approximation $x_0^{(0)}$, we compute successive approximations to the solution of $C x_0 = w$ using the following asynchronous Jacobi-like iteration scheme

$$
x_k^{(k+1)} = D_{00}^{-1} (w - C x_0^{(k)}) + x_0^{(k)}
$$

or

$$
x_k^{(k+1)} = r_k^{(k)} / d_{00i} + x_0^{(k)}, \quad i = 1, \ldots, n_0
$$

where the subscript $i$ denotes the $i$-th component of a vector and $r_k^{(k)} = w - C x_0^{(k)}$ is the vector of residuals for the $k$-th iteration. In its implementation we use the relative norm of the residual as the stopping criterion.

In order to study the convergence of the iterative scheme (3.6), we carried out several experiments on sequential machines. Table 3.1 summarizes the results of these experiments for the PDE problem.

$$
Lu = u_{xx} + u_{xy} + u_{yy} + u_x + u_y + u \quad \text{on} \quad \Omega = [0,1] \times [0,1]
$$

$$
Bu = u \quad \text{on} \quad \partial \Omega
$$

(3.7)
Table 3.1. Number of iterations required to reduce the relative residual to $\varepsilon = 10^{-6}$ for the problem (3.7), with several grid sizes and domain decompositions.

<table>
<thead>
<tr>
<th>Grid Dimensions</th>
<th>Domain Decomposition</th>
<th>Size of C</th>
<th>Iterations $\omega = 1$</th>
<th>Optimal $\omega$</th>
<th>Iterations $\omega = \text{optimal}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>$N$</td>
<td>$MP$</td>
<td>$NP$</td>
<td>$P$</td>
<td>$n_0$</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>33</td>
<td>161</td>
<td>0.50</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>99</td>
<td>284</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>65</td>
<td>302</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>99</td>
<td>248</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>231</td>
<td>466</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>129</td>
<td>359</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>129</td>
<td>366</td>
<td>0.85</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>231</td>
<td>435</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>16</td>
<td>495</td>
<td>789</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>257</td>
<td>505</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>16</td>
<td>189</td>
<td>422</td>
<td>0.85</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>16</td>
<td>257</td>
<td>513</td>
<td>0.85</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>16</td>
<td>495</td>
<td>733</td>
<td>0.50</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>49</td>
<td>250</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>147</td>
<td>413</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>97</td>
<td>432</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>147</td>
<td>369</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>343</td>
<td>667</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>193</td>
<td>521</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>193</td>
<td>526</td>
<td>0.85</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>343</td>
<td>621</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>16</td>
<td>735</td>
<td>1001</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>385</td>
<td>731</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>16</td>
<td>285</td>
<td>603</td>
<td>0.85</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>16</td>
<td>385</td>
<td>731</td>
<td>0.85</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>16</td>
<td>735</td>
<td>1001</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Figure 3.3. Graph of the $n_{iter}$ vs $n_0$ with $\omega = 1$ and $\omega$ equal to the computed optimal.
where the true solution is \( u = x^{15/2} \cdot y^{13/2} \). These numerical experiments suggest that the number of iterations needed to reduce the relative residual to \( \varepsilon = 10^{-6} \) grows linearly with the size of \( C \), i.e., with \( M \cdot NP + N \cdot MP \). In fact, Table 3.1 suggests that the average number of iterations is \( 3n_0 \). It is worth noticing that for a constant number of processors the least number of iterations is obtained for \( NP = MP \). This is in agreement with the observations of others studying domain decomposition. There is, though, the possibility of reducing significantly the number of iterations by introducing a relaxation factor \( \omega \) in the iteration formula, thus transforming it into an accelerated Jacobi-like iteration

\[
x^{(k+1)}_0 = D_{00}^{-1} (g - x^{(k)}_0) / \omega + x^{(k)}_0.
\]

The last column of Table 3.1 shows the number of iterations required for problem (3.7) using various values of \( \omega \). Figure 3.3 shows also the growth of \( n_{iter} \) with the size of \( C \). Additional experiments have shown that \( n_{iter} \) does not depend much on the nature of \( u \) and operator \( L \). Also the optimal value \( \omega \) was not affected by the size of the grid and \( u \), but it appears that it depends on \( L \).

4. THE PARALLEL QUADRATIC SPLINE COLLOCATION–CAPACITANCE METHOD

In this section, we define the parallel quadratic spline collocation–capacitance method (PQSCC method) and discuss its implementation and complexity. Assume we have \( P \) processors. Each processor is assigned to handle the computations associated with \( n_0/P \) rows of \( A_{00} \) and \( A_{01} \). In case of the remaining rows, the last \( n_0 - \lfloor n_0/P \rfloor \cdot P \) processors are assigned one additional row of \( A_{00} \) and \( A_{01} \). Each processor is also assigned to handle the computations of one block of \( A_{11} \) and the respective rows of \( A_{10} \). We will assume for simplicity that \( MP \) divides \( M+2 \) and \( NP \) divides \( N+2 \). This partitions the matrices \( A_{00}^p, A_{01}^p \) and \( A_{10}^p \), assigned to the \( p \)th processor for \( p = 1, \ldots, P \). The matrices \( A_{00}^p, A_{01}^p \) and \( A_{10}^p \) are stored in sparse matrix form, while respective rows of \( A_{11}^p \) are stored in LINPACK band form. All of them are stored in the local memory of processor \( p \).

Specifically, the algorithm for the \( p \)th processor written in a pseudo-language consists of the following statements:

**Code executed by the \( p \)th processor**

01. Solve \( A_{11}^p x^p = b^p \)
02. Distribute \( x^p \) among all other processors
03. Receive \( x_q^p \) from all other processors, \( q \neq p \), and update \( x_1 \).
04. Compute \( g^p = b^p - A_{00}^p x_1 \)
05. \( k = 0 \)
06. Compute initial guess \( x_0^{(0)} \) for \( x_0 \)
07. for \( k = 0, \ldots, \text{maxit} \) do /* maxit is the maximum number of iterations allowed */
08. Compute \( A_0^k x_0^{(k)} \)
09. Solve \( A_1^k t^p = -A_0^k x_0^{(k)} \) for \( t^p \)
10. Distribute \( t^p \) among all other processors
11. Receive \( t^p \) from all other processors, \( q \neq p \), and store it in \( t \)
12. Compute \( A_0^k x_0^{(k)} + A_1^k t \) /* this is \( C^p x_0 \) */
13. Compute residual \( r^p = g^p - C^p x_0 \)
14. If \( |r^p| \leq \text{eps} \) send satisfaction flag to other processors
15. else send continuation flag /* \( \text{eps} \) is the precision required */
16. Receive flags from all other processors
17. If all flags are satisfactory exit loop
18. Update \( x_0^k, x_0^{(k+1)} = D_0^k r^p + x_0^{(k)} \)
19. \( k = k+1 \)
20. Distribute \( x_0^{(k)} \) among all other processors
21. Receive \( x_0^{(k)} \) from all other processors, \( q \neq p \), and store it in \( x_0 \)
22. endfor
23. Distribute final \( x_0 \) among all other processors
24. Receive \( x_0 \) from all other processors, \( q \neq p \), and store it in \( x_0 \).
25. Compute \( A_0^k x_0 \)
26. Solve \( A_1^k x_1^p = b^p - A_0^k x_0 \)
27. Send final \( x_0 \) and \( x_1^p \) to host processor

To measure the processing time of the PQSCC method, we counted the operations needed for each computational step and also carried out several numerical experiments. With the grid size \((M \times N)\), the number of processors \((P)\) and the domain decomposition \((MP \times NP = P)\), the complexity of the algorithm is summarized in the following lemma. Time is measured in units of one arithmetic operation, communication is assumed to be instantaneous.

**Lemma 2.** Assuming an \( MP \times NP \) decomposition of domain \( \Omega \), then the processing time required to solve the spline collocation equations (2.3) on a \( P \)-processor MIMD machine with the PQSCC method is

\[
O\left(\frac{M^3 \cdot N}{MP^2 \cdot P}\right) + \text{niter} O\left(\frac{M^2 \cdot N}{MP \cdot P}\right)
\]

(3.7)

where \( \text{niter} \) is the number of iterations of the Jacobi scheme (3.6) for solving the corresponding capacitance system.

**Proof:** Based on the previously presented algorithm that each processor executes, it is clear that the steps that dominate in processing time are Steps 1 and 9. Note that Step 1 is executed only once and the factorization of the blocks \( A_1^p, \, p = 1, \ldots, P \) is saved, while Step 9 is executed \( \text{niter} \) times. The factorization of each block takes

\[
O\left(\left(\frac{M+3}{MP}\right)^2 \left(\frac{M+3}{MP} - 1\right) \left(\frac{N+3}{NP} - 1\right)\right)
\]

time which gives the first component of (3.7), and the
back substitution $O\left(\frac{M+3}{MP}\right)\frac{M+3}{MP-1}\left(\frac{N+3}{NP}-1\right)$ time which gives the second component of (3.7).

Based on our experiments reported in Section 3.3 (Table 3.1 and Figure 3.3), we can safely assume that $\text{niter} = O(M \cdot NP + N \cdot MP)$. Suppose for simplicity that $M = N$, then the complexity of the PQSCC method is of the order $O\left(M^4\frac{1}{(MP^2 + \frac{1}{P})}\right)$ which is the same order as the time required to solve the capacitance system. From the complexity of the PQSCC method, we make the following important observation, which we formulate as a corollary.

Corollary 2. Assuming the number of iterations required by the Jacobi scheme (3.6) to reduce the residual $r = w - Cx_0$ to $\varepsilon$ grows linearly with $M \cdot NP + N \cdot MP$, then the optimal PQSCC implementation is based on a domain decomposition consisting of vertical strips, i.e., $MP = P$ and $NP = 1$.

The Corollary 2 is a consequence of the fact that the bandwidth of $A_{ij}^1$ is $O(M/MP)$. The above observations are supported by the numerical data of Tables 4.1a, 4.1b and 4.1c. All computations in these tables were carried out in single precision on a NCUBE/7 hypercube machine with 128 processors and convergence tolerance $\varepsilon = 10^{-5}$. 
Table 4.1a. Timing of PQSCC on the NCUBE/7 in msecs for different domain decompositions and grid sizes. The total time of the algorithm is presented as a sum of the required discretization and solution times. The iteration time includes processing and communication time per iteration.

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>MP</th>
<th>NP</th>
<th>P</th>
<th>NIT</th>
<th>TOTAL TIME</th>
<th>ITER TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>26</td>
<td>230</td>
<td>259 + 341</td>
<td>17.63</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>19</td>
<td>255 + 351</td>
<td>15.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>150 + 415</td>
<td>12.65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>2</td>
<td>1094</td>
<td>1924*</td>
<td>97.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>77</td>
<td>1211 + 8329</td>
<td>72.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>54</td>
<td>1172 + 4277</td>
<td>41.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>91</td>
<td>651 + 5070</td>
<td>37.56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>96</td>
<td>618 + 4214</td>
<td>37.56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>76</td>
<td>626 + 2996</td>
<td>37.56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>114</td>
<td>339 + 3357</td>
<td>28.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>113</td>
<td>340 + 3036</td>
<td>22.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>130</td>
<td>198 + 2944</td>
<td>22.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>1</td>
<td>2600</td>
<td>8459*</td>
<td>59.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>124</td>
<td>2873 + 4032</td>
<td>19.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>88</td>
<td>2832 + 18980</td>
<td>19.17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>153</td>
<td>1557 + 24156</td>
<td>147.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>160</td>
<td>1460 + 17835</td>
<td>107.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>129</td>
<td>1462 + 12529</td>
<td>93.76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>246</td>
<td>717 + 18555</td>
<td>73.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>195</td>
<td>780 + 12502</td>
<td>62.34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>196</td>
<td>769 + 11169</td>
<td>55.86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>218</td>
<td>773 + 11650</td>
<td>52.63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>286</td>
<td>426 + 11842</td>
<td>40.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>226</td>
<td>497 + 8879</td>
<td>38.73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>267</td>
<td>422 + 10092</td>
<td>37.38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>32</td>
<td>303</td>
<td>248 + 9745</td>
<td>31.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>301</td>
<td>254 + 9531</td>
<td>31.39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>64</td>
<td>362</td>
<td>167 + 11249</td>
<td>30.88</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Time to solve equations (2.3) by Band GE NO PIV [13]
Table 4.1b. Timing of PQSCC on the NCUBE/7 in msecs for different domain decompositions and grid sizes. The total time of the algorithm is presented as a sum of the required discretization and solution times.

<table>
<thead>
<tr>
<th>$M$</th>
<th>$N$</th>
<th>$MP$</th>
<th>$NP$</th>
<th>$P$</th>
<th>$NITER$</th>
<th>TOTAL TIME</th>
<th>ITER TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>171</td>
<td>4744 + 25502*</td>
<td>664.50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>122</td>
<td>5609 + 124679</td>
<td>442.58</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>212</td>
<td>2901 + 74031</td>
<td>326.26</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>223</td>
<td>2692 + 51097</td>
<td>220.76</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>181</td>
<td>2662 + 30687</td>
<td>164.55</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>352</td>
<td>1500 + 56954</td>
<td>156.84</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>275</td>
<td>1418 + 34092</td>
<td>120.77</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>278</td>
<td>1384 + 26750</td>
<td>94.61</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>316</td>
<td>1381 + 27109</td>
<td>84.80</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>558</td>
<td>793 + 40434</td>
<td>72.01</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>403</td>
<td>759 + 29183</td>
<td>71.51</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>320</td>
<td>753 + 20040</td>
<td>61.87</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>386</td>
<td>740 + 22170</td>
<td>56.97</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>486</td>
<td>752 + 28876</td>
<td>59.05</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>32</td>
<td>2</td>
<td>600</td>
<td>426 + 29163</td>
<td>48.41</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>435</td>
<td>423 + 20652</td>
<td>47.18</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>428</td>
<td>421 + 19350</td>
<td>44.92</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>541</td>
<td>423 + 24913</td>
<td>45.84</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2</td>
<td>650</td>
<td>255 + 27244</td>
<td>41.80</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>528</td>
<td>257 + 21760</td>
<td>41.08</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>570</td>
<td>258 + 23686</td>
<td>41.43</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>128</td>
<td>2</td>
<td>710</td>
<td>181 + 30096</td>
<td>42.29</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>627</td>
<td>183 + 26607</td>
<td>42.30</td>
<td></td>
</tr>
</tbody>
</table>

* Time to solve (2.3) by Band GE NO PIV [13]
Table 4.1c. Timing of PQSCC on the NCUBE/7 in msecs for different domain decompositions and grid sizes. The total time of the algorithm is presented as a sum of the required discretization and solution times.

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>MP</th>
<th>NP</th>
<th>P</th>
<th>NIT</th>
<th>TOTAL TIME</th>
<th>ITER TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>38</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8000 + 60508*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>200</td>
<td>8514 + 155892</td>
<td>770.12</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>271</td>
<td>4793 + 178938</td>
<td>615.22</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>286</td>
<td>4336 + 117400</td>
<td>396.04</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>232</td>
<td>4240 + 72449</td>
<td>303.97</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>452</td>
<td>2493 + 137960</td>
<td>293.82</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>354</td>
<td>2266 + 76474</td>
<td>210.52</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>358</td>
<td>2191 + 60855</td>
<td>167.35</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>413</td>
<td>2173 + 57990</td>
<td>139.07</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>4</td>
<td>8</td>
<td>519</td>
<td>1200 + 62087</td>
<td>118.06</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>415</td>
<td>1179 + 42727</td>
<td>101.76</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>500</td>
<td>1154 + 44393</td>
<td>88.21</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>32</td>
<td>16</td>
<td>16</td>
<td>526</td>
<td>955 + 37724</td>
<td>71.27</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>32</td>
<td>16</td>
<td>32</td>
<td>810</td>
<td>550 + 47662</td>
<td>58.64</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>64</td>
<td>8</td>
<td>16</td>
<td>564</td>
<td>655 + 39899</td>
<td>70.31</td>
</tr>
</tbody>
</table>

5. MAPPING PQSCC TO A HYPERCUBE ARCHITECTURE

In this section, we study the mapping of the PQSCC algorithm to a hypercube architecture and discuss its implementation on the NCUBE/7 with 128 processors. One of the primary objectives of the mapping process is the minimization of the communication cost. According to the description of the algorithm in Section 4, the processors need to exchange the parts of the solution that each computes. For example, in the case of vector \( x_0 \) (length \( n_0 \)), each processor computes \( n_0/P \) of its components and sends them to every other processor. This implies that each \( x_0 \) processor generates \( P \) messages of size \( n_0/P \). Assuming no overlap of communication and that the cost of each message is proportional to its size and the length of the path between processors plus an initialization overhead, then we conclude that the total communication cost for completing the update of \( x_0 \) is \( O(P \cdot A + n_0B \log P) \) where \( A \) is the overhead constant and \( B \) the average cost per message.

Following Stout [12], we have implemented a faster communication scheme for updating the solution vector on all processors where messages are grouped to produce bigger messages which in turn can be broadcast in less time. In this scheme, first the processors exchange vector parts with their lowest bit neighbor and update the corresponding parts of the solution vector. Then they exchange vector parts with the second lowest bit neighbor. This exchange of data continues until they reach to the highest bit neighbor.
The code executed by each processor to exchange data with all other processors follows. It is written in a pseudo language extended with the operations `send` (message, processor) and `get` (message, processor), which respectively perform a send of the message to the processor and a receive of a message from a processor.

```plaintext
for bit = 1, log P do
    send (my_data, bit_neighbor_processor)
    get (others_data, bit_neighbor_processor)
    my_data = my_data and others_data
end for
```

Initially each processor has its own computed data in `my_data`. Finally `my_data` contains the data from all processors. The operation `and` means either concatenating two vectors, or doing logical operations with flags.

Figure 5.1 shows graphically the steps of the exchange procedure for a third order cube where each processor has computed 10 components of a vector. Assuming the previously defined notation, each processor generates $\log P$ messages of increasing size and the total communication cost is $O(A \log P + Bn_0)$. This is a consequence of the fact that the total length of the messages generated by one processor is

$$\frac{n_0 \log P - 1}{P} \sum_{i=0}^{\log P - 1} 2^i = n_0 \frac{P - 1}{P}.$$  

The above scheme turns out to be optimal, since the longest path among processors on a hypercube architecture is of order $\log P$. 
6. NUMERICAL RESULTS

We have implemented the PQSCC algorithm on several configurations of the NCUBE/7 hypercube machine and measured its performance for various grids. Figures 6.1 and 6.2 show these data in graphical form. Several quantitative data are given in Tables 4.1a,b,c. In Figure 6.3, we compare the performance of the PQSCC to sequential Gauss elimination (Band GE NO PIV [13]). These data indicate that the efficiency of the method depends on the size of the problem. This is true for any MIMD algorithm. Given the memory limitations of the current NCUBE configuration, one can not achieve a balance between the processing and communication time for a large number of processors. Although we have found a relatively fast way to solve the capacitance system (3.2), we feel that there is potential for further improvement. We are currently studying different alternatives.
Figure 6.1. Timing of PQSCC algorithm in ms on the NCUBE/7 with the acceleration constant $\omega$ equal to the computed optimal one. In this graph we plot the total time (solid line) and processing time (dotted line) versus the number of NCUBE processors (nodes) used for additional grids. The accelerated Jacobi method (3.6) was used to solve the capacitance system with tolerance $\varepsilon = 10^{-6}$. The dots on the left indicate the processing time of Band GE without pivoting to solve the same collocation equations.
Figure 6.2. This graph is similar to the one in Figure 6.1 with $\omega = 1$ as the acceleration constant.
Figure 6.3 This graph corresponds to Figure 6.2 and plots the speedup vs number of processors. The speedup is determined with respect to the time of the PQSCC algorithm on two processors.
REFERENCES


