Purdue University
Purdue e-Pubs

Department of Computer Science Technical Reports | Department of Computer Science

1988

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

Christara, C. C.; Houstis, Elias N.; and Rice, John R., "A Parallel Spline Collocation-Capacitance Method for Elliptic Parallel Differential Equations" (1988). *Department of Computer Science Technical Reports.* Paper 634.
https://docs.lib.purdue.edu/cstech/634

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

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 49707

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)

$$Lu \equiv aD_x^2u + bD_xD_yu + cD_y^2u + dD_xu + eD_yu + fu = g \quad \text{in } \Omega \tag{1.1}$$

subject to mixed type boundary conditions

$$Bu \equiv \alpha u + \beta D_n u = g_0 \qquad \text{on } \partial\Omega$$

where $\Omega \equiv [ax, bx] \times [ay, by]$ 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_nu is the normal derivative of u and $\partial\Omega$ is the boundary of Ω.

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 $\Delta_x \equiv \{x_k = ax + kh_x;\ k = 0, \ldots, M\}$, with $h_x = \dfrac{bx - ax}{M}$ be a uniform partition of the interval $[ax, bx]$ and $\Delta_y \equiv \{y_l = ay + lh_y;\ l = 0, \ldots, N\}$, with $h_y = \dfrac{by - ay}{N}$ a uniform partition of $[ay, by]$. Throughout we denote by $\Delta \equiv \Delta_x \times \Delta_y$, the induced grid partition of Ω and by τ_i^x, $i = 1..M$ the midpoints of Δ_x and by τ_j^y, $j = 1, \ldots, N$ the midpoints of Δ_y. For convenience we extend the notation so that $\tau_0^x \equiv x_0$, $\tau_{M+1}^x \equiv x_M$, $\tau_0^y \equiv y_0$, $\tau_{N+1}^y \equiv y_N$. For later use we define the following sets of points: $T \equiv \{(\tau_i^x, \tau_j^y),\ i = 0, \ldots, M+1,\ j = 0, \ldots, N+1\}$ the set of *collocation points*, $T_i \equiv \{(\tau_i^x, \tau_j^y),\ i = 2, \ldots, M-1,\ j = 2, \ldots, N-1 \subset T$ the subset of *interior collocation points*, $T_{ic} \equiv \{(\tau_1^x, \tau_1^y), (\tau_M^x, \tau_1^y), (\tau_1^x, \tau_N^n), (\tau_M^x, \tau_N^y)\} \subset T$ the set of *corner collocation points* of Ω , $T_b \equiv T \cap \partial\Omega$ the set of *boundary collocation points* in T and $T_{ib} \equiv T - (T_i \cup T_{ic} \cup T_b)$ 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 $S_{2,\Delta} \equiv P_{2,\Delta} \cap C^1(\Omega)$, where $P_{2,\Delta}$ is the tensor product of one-dimensional piecewise quadratic polynomials in x and y over the partitions Δ_x and Δ_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

$$\frac{1}{24}\begin{bmatrix} & & c\,D_y^2 & S_{i,j+1} & \\ & & -b\,D_x\,D_y & S_{i,j+1} & \\ & & -e\,D_y & S_{i,j+1} & \\ & & & & \\ & & -2a\,D_x^2 & S_{i,j} & \\ a\,D_x^2 & S_{i-1,j} & -4b\,D_x\,D_y & S_{i,j} & a\,D_x^2 \quad S_{i+1,j} \\ -b\,D_x\,D_y & S_{i-1,j} & -2c\,D_y^2 & S_{i,j} & -b\,D_x\,D_y\,S_{i+1,j} \\ -d\,D_x & S_{i-1,j} & +2d\,D_x & S_{i,j} & -d\,D_x \quad S_{i+1,j} \\ & & +2e\,D_y & S_{i,j} & \\ & & & & \\ & & c\,D_y & S_{i,j-1} & \\ & & -b\,D_x\,D_y & S_{i,j-1} & \\ & & -e\,D_y & S_{i,j-1} & \end{bmatrix}$$

For each interior-boundary collocation point on $x = \tau_1^x$, $P_L S$ is defined by the stencil

$$\frac{1}{24}\begin{Bmatrix}
-bD_xD_y & S_{1,j+1} & & & & & & & & \\
+cD_y^2 & S_{1,j+1} & & & & & & & & \\
-eD_y & S_{1,j+1} & & & & & & & & \\
 & & & & & & & & & \\
+2aD_x^2 & S_{1,j} & -5a\,D_x^2 & S_{2,j} & +4a\,D_x^2 & S_{3,j} & -a\,D_x^2 & S_{4,j} \\
 & & +5b\,D_x\,D_y & S_{2,j} & -4b\,D_x\,D_y & S_{3,j} & +b\,D_x\,D_y & S_{4,j} \\
-2cD_y^2 & S_{1,j} & & & & & & & & \\
-2dD_x & S_{1,j} & +5d\,D_x & S_{2,j} & -4d\,D_x & S_{3,j} & +d\,D_x & S_{4,j} \\
+2eD_y & S_{1,j} & & & & & & & & \\
 & & & & & & & & & \\
-bD_xD_y & S_{1,j-1} & & & & & & & & \\
+cD_y^2 & S_{1,j-1} & & & & & & & & \\
-eD_y & S_{1,j-1} & & & & & & & &
\end{Bmatrix}.$$

Then P_LS is defined by similar stencils at the rest of the interior boundary collocation points in T_{ib} corresponding to $x=\tau_M^x$, $y=\tau_1^y$ and $y=\tau_N^y$. Further, P_LS is defined at the corner collocation point (τ_1^x, τ_1^y) by the stencil

$$\frac{1}{24}\begin{Bmatrix}
+b\,D_x\,D_y & S_{1,4} & & & & & & & & \\
-c\,D_y^2 & S_{1,4} & & & & & & & & \\
+e\,D_y & S_{1,4} & & & & & & & & \\
 & & & & & & & & & \\
-4b\,D_x\,D_y & S_{1,3} & & & & & & & & \\
+4c\,D_y^2 & S_{1,3} & & & & & & & & \\
+4e\,D_y & S_{1,3} & & & & & & & & \\
 & & & & & & & & & \\
+5b\,D_x\,D_y & S_{1,2} & & & & & & & & \\
-5c\,D_y^2 & S_{1,2} & & & & & & & & \\
+5e\,D_y & S_{1,2} & & & & & & & & \\
 & & & & & & & & & \\
+2a\,D_x^2 & S_{1,1} & +5a\,D_x^2 & S_{2,1} & -4a\,D_x^2 & S_{3,1} & -a\,D_x^2 & S_{4,1} \\
-4b\,D_x\,D_y & S_{1,1} & +5b\,D_x\,D_y & S_{2,1} & -4b\,D_x\,D_y & S_{3,1} & +b\,D_x\,D_y & S_{4,1} \\
+2c\,D_y^2 & S_{1,1} & & & & & & & & \\
-2d\,D_x & S_{1,1} & +5d\,D_x & S_{2,1} & -4d\,D_x & S_{3,1} & +d\,D_x & S_{4,1} \\
-2e\,D_y & S_{1,1} & & & & & & & &
\end{Bmatrix}.$$

Similar stencils define P_LS at the rest of the corner collocation points in T_{ic}. 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

$$\begin{aligned} LS &= g + O(h^2) \quad \text{on } T - T_b, \\ BS &= g_0 + O(h^2) \quad \text{on } T_b \end{aligned} \tag{2.1}$$

and

$$\begin{aligned} L'S &= g + O(h^4) \text{ or } LS = g - P_L S + O(h^4) \text{ on } T - T_b, \\ B'S &= g_0 + O(h^4) \text{ or } BS = g_0 - P_B S + O(h^4) \text{ on } T_b \end{aligned} \tag{2.2}$$

where $L'S \equiv LS + P_L S$, $B'S \equiv BS + P_B S$, $P_L S$ 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_Δ to the true solution u of (1.1) is forced to satisfy

$$\text{P2C1COL:} \qquad \begin{aligned} Lu_\Delta &= g \quad \text{on } T - T_b, \\ Bu_\Delta &= g_0 \quad \text{on } T_b. \end{aligned} \tag{2.3}$$

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

P2C1CL1:
$$\begin{aligned} L'u_\Delta &= g \quad \text{on } T - T_b, \\ B'u_\Delta &= g_0 \quad \text{on } T_b \end{aligned} \tag{2.4}$$

A mathematically equivalent but computationally advantageous version of this method is the *two step deferred correction method* defined by

P2C1CL2: (1st step)
$$\begin{aligned} Lv &= g \quad \text{on } T - T_b, \\ Bv &= g_0 \quad \text{on } T_b, \end{aligned} \tag{2.5a}$$

(2nd step)
$$\begin{aligned} Lu_\Delta &= g - P_L v \quad \text{on } T - T_b, \\ Bu_\Delta &= g_0 - P_B v \quad \text{on } T_b. \end{aligned} \tag{2.5b}$$

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 $5N + 11$.

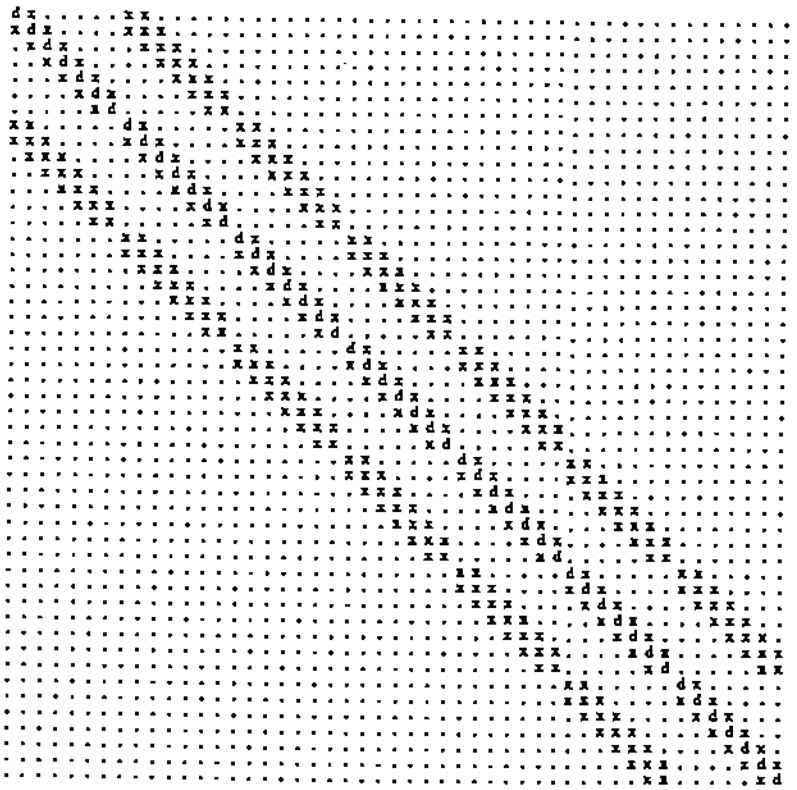


Figure 2.1. Structure of the matrix of collocation equation corresponding to P2C1COL for $N = M = 5$. x denotes a non-zero off diagonal element, d a non-zero diagonal one, while all zero entries are represented by "." character.

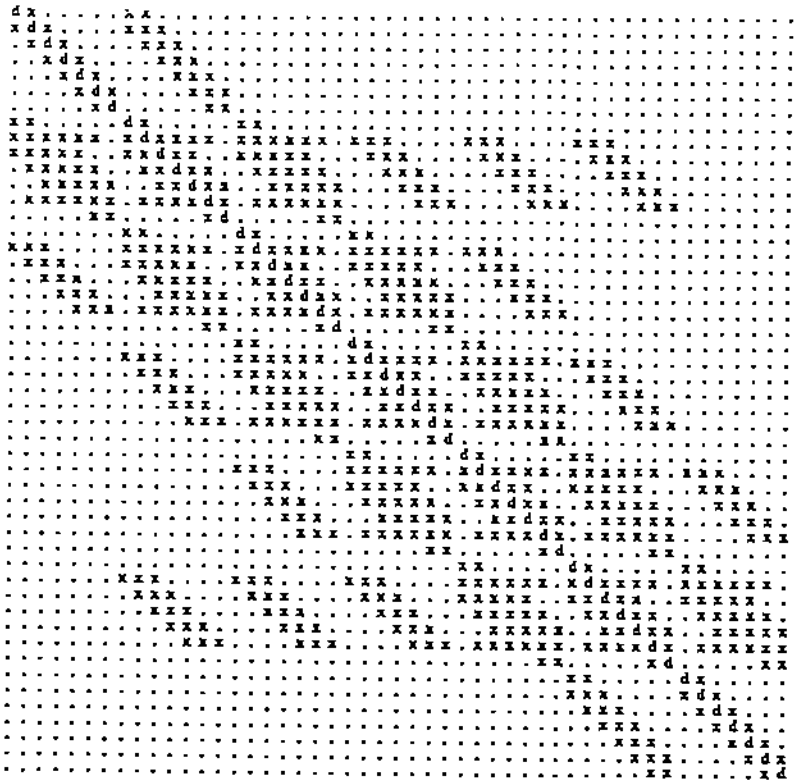


Figure 2.2. Structure of the matrix of collocation equations corresponding to P2C1CL1 for $N = M = 5$. The notation of Figure 2.1 is used here.

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

$$\begin{aligned} Lu &\equiv (e^{xy}\, u_x)_x + (e^{-xy}\, u_y)_y - \frac{u}{1+x+y} \quad \text{in } \Omega \equiv [0,1]\times[0,1], \\ Bu &\equiv u \qquad\qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega \end{aligned} \tag{2.6}$$

whose true solution is $u = 0.75e^{xy}\sin(\pi x)\sin(\pi y)$. 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 29×29 uniform grid.

Method Solver*	P2C1CL1	P2C1CL2
SSOR SI	14.3	5.4
Envelope LDU	22.0	4.3
Band GE No Piv	54.3	4.9
Sparse GE No Piv	33.8	6.2
Linpack Band	75.0	5.3

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

Method	P2C1CL1	P2C1CL2	Galerkin(2,1)	Galerkin(3,2)
Grid				
5×5	0.167	0.105	0.147	0.351
9×9	0.662	0.324	0.564	1.366
17×17	4.034	1.660	2.913	7.332
33×33	39.370	13.715	22.602	54.683

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.
Sparse GE No Piv: An LU factorization of a matrix using a fast storage conserving non-symmetric scheme.
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.

Method	P2C1COL		P2C1CL1		P2C1CL2		Galerkin (2.1)	
Grid	Error	Convergence	Error	Convergence	Error	Convergence	Error	Convergence
5×5	2.4–2		2.7–3		2.8–3		5.9–3	
		1.98		3.32		3.32		4.25
9×9	6.1–3		2.7–4		2.8–4		3.1–4	
		2.02		3.91		3.81		4.11
17×17	1.5–3		1.8–5		2.0–5		1.8–5	
		2.10		3.91		3.94		4.03
33×33	3.5–4		1.2–6		1.3–6		1.1–6	

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

$$A_{00}\, x_0 + A_{01}\, x_1 = b_0 \tag{3.2a}$$

$$A_{10}\, x_0 + A_{11}\, x_1 = b_1 \tag{3.2b}$$

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 Ω 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 Ω that leads to the above partition of the system is that involving two vertical strips, say Ω_1, Ω_2, where x_0 is the vector of unknowns that belong to the middle line that separates Ω_1 from Ω_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

$$Cx_0 \equiv (A_{00} - A_{01}\, A_{11}^{-1}\, A_{10})\, x_0 = b_0 - A_{01}\, A_{11}^{-1}\, b_1 \equiv w. \tag{3.3}$$

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

$$A_{11}\, x_1 = b_1 - A_{10}\, x_0. \tag{3.4}$$

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 Ω in $P \equiv MP \times NP$ 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 int 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:

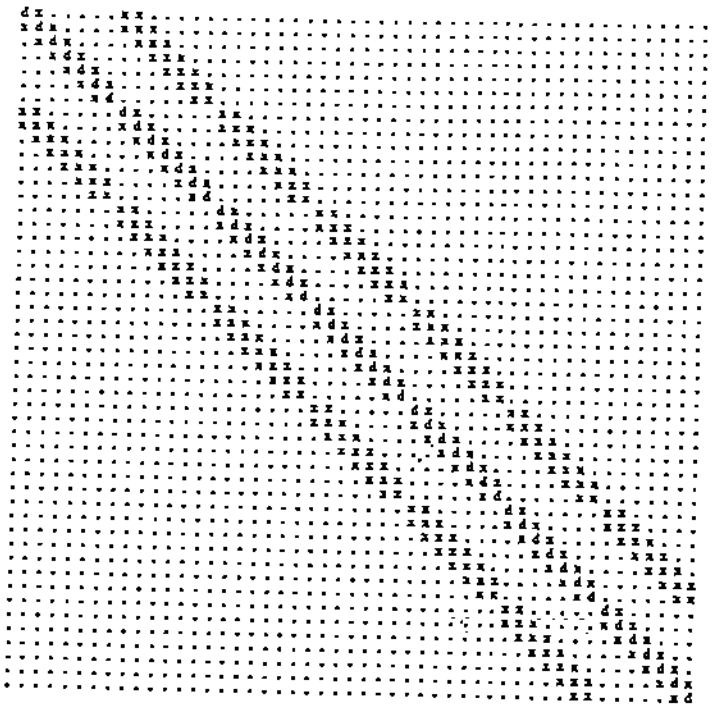


Figure 3.1(a). Structure of the matrix of collocation equations (2.3) for $N = M = 5$ grid with the original ordering. d, x, $\cdot$ are defined as in Figure 2.1.

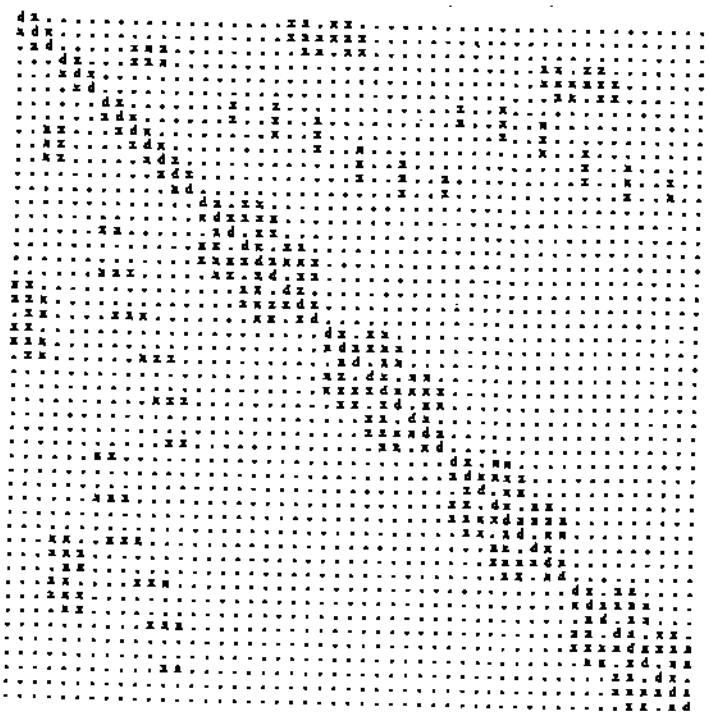


Figure 3.1(b). Structure of the matrix of collocation equations for $N = M = 5$ grid and a 2×2 domain decomposition renumbering. d, x, $\cdot$ are defined as in Figure 2.1.

1. Solve $A_{11}\,\tilde{x}_1 = b_1$,
2. Compute $w = b_0 - A_{01}\,\tilde{x}_1$,
3. Solve $Cx_0 = w$, $C \equiv A_{00} - A_{01}\,A_{11}^{-1}\,A_{10}$,
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 $A_{11}\,\tilde{x}_1 = b_1$ is equivalent to solving $A_{11}^p\,\tilde{x}_1^p = b_1^p$, 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 $Cx_0 = w$ 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 $Cx_0 = w$ 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 $Cx_0 = w$ is equivalently written in the form

$$D_{00}\,x_0 = w - Cx_0 + D_{00}\,x_0\,. \tag{3.5}$$

Starting with some initial approximation $x_0^{(0)}$, we compute successive approximations to the solution of $Cx_0 = w$ using the following asynchronous Jacobi-like iteration scheme

$$x_0^{(k+1)} = D_{00}^{-1}\,(w - Cx_0^{(k)}) + x_0^{(k)}$$

or

$$x_{0i}^{(k+1)} = r_i^{(k)}/d_{00i} + x_{0i}^{(k)},\quad i = 1, \ldots, n_0 \tag{3.6}$$

where the subscript i denotes the i-th component of a vector and $r^{(k)} \equiv w - Cx_0^{(k)}$ 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.

$$\begin{aligned} Lu \equiv\ & u_{xx} + u_{xy} + u_{yy} + u_x + u_y + u \quad \text{on } \Omega = [0,1] \times [0,1] \\ Bu \equiv\ & u \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega \end{aligned} \tag{3.7}$$

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.

Grid		Domain Decomposition			Size of C $\omega = 1$	Iterations $\omega = 1$	Optimal ω	Iterations ω = optimal
M	N	MP	NP	P	n_0	*niter*	ω	*niter* ω
32	32	1	2	2	33	222	0.85	190
		2	1	2	33	161	0.50	85
		1	4	4	99	284	0.85	245
		2	2	4	65	302	0.85	258
		4	1	4	99	248	0.50	125
		1	8	8	231	466	0.85	386
		2	4	8	129	359	0.85	317
		4	2	8	129	366	0.85	305
		8	1	8	231	435	0.50	231
		1	16	16	495	789	0.85	656
		2	8	16	257	505	0.85	437
		4	4	16	189	422	0.85	368
		8	2	16	257	513	0.85	436
		16	1	16	495	733	0.50	436
48	48	1	2	2	49	324	0.85	286
		2	1	2	49	250	0.50	129
		1	4	4	147	413	0.85	354
		2	2	4	97	432	0.85	369
		4	1	4	147	369	0.50	197
		1	8	8	343	667	0.85	573
		2	4	8	193	521	0.85	450
		4	2	8	193	526	0.85	444
		8	1	8	343	621	0.50	324
		1	16	16	735	1001	0.85	944
		2	8	16	385	731	0.85	630
		4	4	16	285	603	0.85	527
		8	2	16	385	731	0.85	598
		16	1	16	735	1001	0.50	553

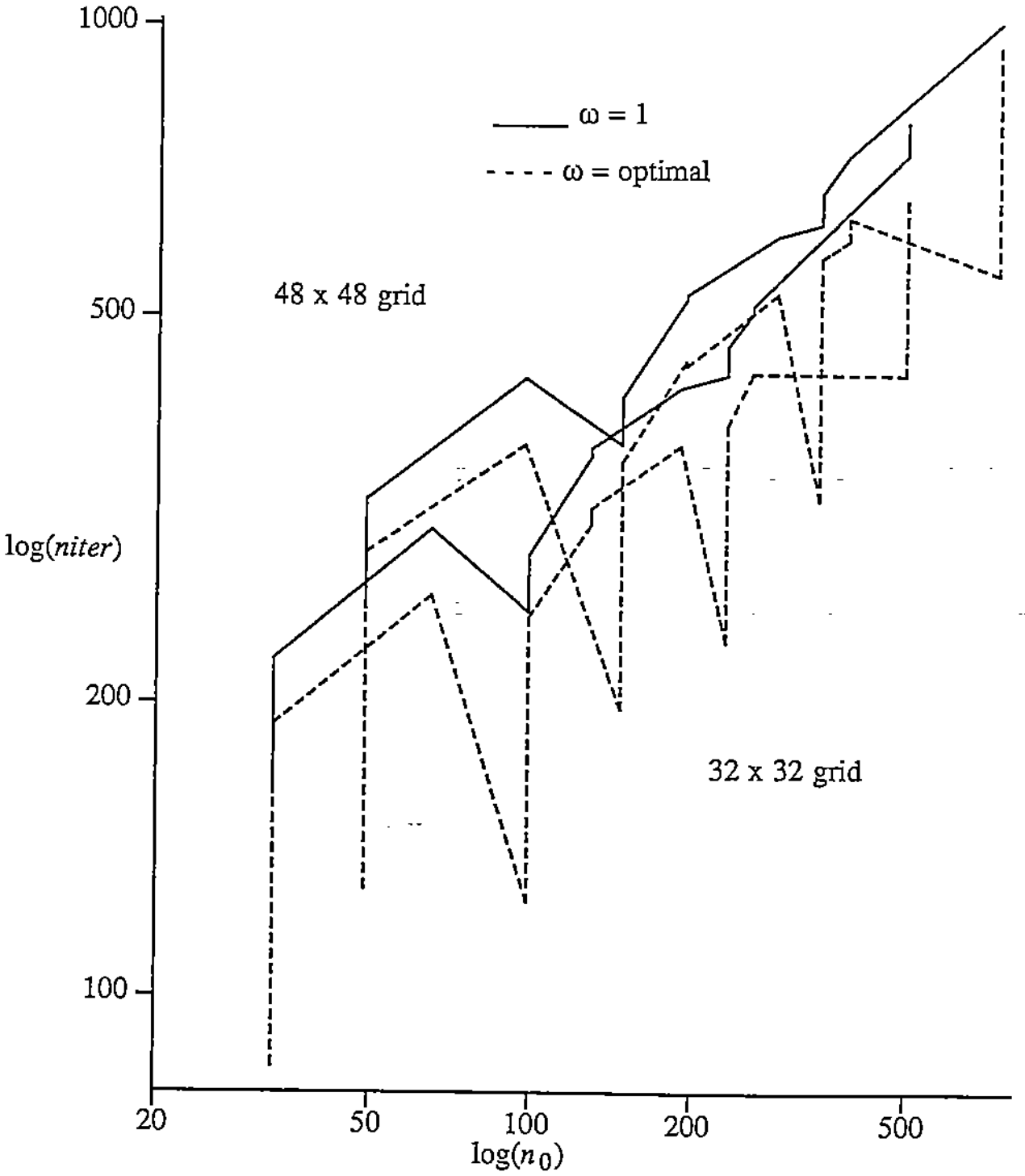


Figure 3.3. Graph of the *niter* vs n_0 with $\omega = 1$ and ω equal to the computed optimal.

where the true solution is $u = x^{13/2} \cdot y^{13/2}$. 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 ω in the iteration formula, thus transforming it into an accelerated Jacobi-like iteration

$$x_0^{(k+1)} = D_{00}^{-1}(g - x_0^{(k)})/\omega + x_0^{(k)}.$$

The last column of Table 3.1 shows the number of iterations required for problem (3.7) using various values of ω. Figure 3.3 shows also the growth of *niter* with the size of C. Additional experiments have shown that *niter* does not depend much on the nature of u and operator L. Also the optimal value ω 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 pth 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 respectives 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 pth processor written in a pseudo-language consists of the following statements:

Code executed by the pth processor

01. Solve $A_{11}^p x_1^p = b_1^p$
02. Distribute x_1^p among all other processors
03. Receive x_1^q from all other processors, $q \neq p$, and update x_1.
04. Compute $g^p = b_0^p - A_{01}^p x_1$
05. $k = 0$
06. Compute initial guess $x_0^{(0)}$ for x_0

07. for $k = 0, \ldots,$ *maxit* do /* maxit is the maximum number of iterations allowed */
08. Compute $A_{10}^p x_0^{(k)}$
09. Solve $A_{11}^p t^p = -A_{10}^p x_0^{(0)}$ for t^p
10. Distribute t^p among all other processors
11. Receive t^q from all other processors, $q \neq p$, and store it in t
12. Compute $A_{00}^p x_0^{(k)} + A_{01}^p t$ /* this is $C^p x_0$ */
13. Compute residual $r^p = g^p - C^p x_0$
14. If $\| r^p \| \leq eps$ send satisfaction flag to other processors
15. else send continuation flag /* *eps* is the precision required */
16. Receive flags from all other processors
17. If all flags are satisfactory exit loop
18. Update x_0^p, $x_0^{p(k+1)} = D_{00}^p r^p + \bar{x}_0^{p(k)}$
19. $k = k+1$
20. Distribute $x_0^{p(k)}$ among all other processors
21. Receive $x_0^{q(k)}$ from all other processors, $q \neq p$, and store it in $x_0^{(k)}$
22. endfor
23. Distribute final x_0^p among all other processors
24. Receive x_0^q from all other processors, $q \neq p$, and store it in x_0.
25. Compute $A_{10}^p x_0^p$
26. Solve $A_{11}^p x_1^p = b_1^p - A_{10}^p x_0$
27. Send final x_0^p 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 Ω, then the processing time required to solve the spline collocation equations (2.3) on a P-processor MIMD machine with the PQSCC method is

$$O\left[\frac{M^3 \cdot N}{MP^2 \cdot P}\right] + niter\ O\left[\frac{M^2 \cdot N}{MP \cdot P}\right] \tag{3.7}$$

where *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_{11}^p, $p = 1, \ldots, P$ is saved, while Step 9 is executed *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 $O\left(\left\{\frac{M+3}{MP}\right\}\left[\frac{M+3}{MP-1}\left\{\frac{N+3}{NP}-1\right\}\right]\right)$ 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 $niter = O(M \cdot NP + N \cdot MP)$. Suppose for simplicity that $M = N$, then the complexity of the PQSCC method is of the order $O\left(M^4\left[\frac{1}{MP^2}+\frac{1}{P}\right]\right)$ 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 ε 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_{11}^{p} 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.

M	*N*	*MP*	*NP*	*P*	*NIT*	TOTAL TIME	ITER TIME
6	6			1		230 + 176*	
		1	2	2	26	259 + 541	17.63
		2	1		19	255 + 351	15.50
		2	2	4		150 + 415	12.65
14	14			1		1094 + 1924*	
		1	2	2	77	1211 + 8329	97.45
		2	1		54	1172 + 4277	72.02
		1	4	4	91	651 + 5070	51.96
		2	2		96	618 + 4214	41.91
		4	1		76	626 + 2996	37.56
		2	4	8	114	339 + 3357	28.48
		4	2		113	340 + 3036	26.07
		4	4	16	130	198 + 2944	22.12
22	22			1		2600 + 8459*	
		1	2	2	124	2873 + 4032	295.48
		2	1		88	2832 + 18980	199.17
		1	4	4	153	1557 + 24156	147.48
		2	2		160	1460 + 17835	107.14
		4	1		129	1462 + 12529	93.76
		1	8	8	246	817 + 18555	73.39
		2	4		195	780 + 12502	62.34
		4	2		196	769 + 11169	55.86
		8	1		218	773 + 11650	52.63
		2	8	16	286	426 + 11842	40.91
		4	4		226	497 + 8879	38.73
		8	2		267	422 + 10092	37.38
		4	8	32	303	248 + 9745	31.86
		8	4		301	254 + 9531	31.39
		8	8	64	362	167 + 11249	30.88

* 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.

M	*N*	*MP*	*NP*	*P*	*NITER*	TOTAL TIME	ITER TIME
30	30			1		4744 + 25502*	
		1	2	2	171	5609 + 124679	664.50
		2	1		122	5266 + 55459	442.58
		1	4	4	212	2901 + 74031	326.26
		2	2		223	2692 + 51097	220.76
		4	1		181	2662 + 30687	164.55
		1	8	8	352	1500 + 56954	156.84
		2	4		275	1418 + 34092	120.77
		4	2		278	1384 + 26750	94.61
		8	1		316	1381 + 27109	84.80
		1	16	16	558	793 + 40434	72.01
		2	8		403	759 + 29183	71.51
		4	4		320	753 + 20040	61.87
		8	2		386	740 + 22170	56.97
		16	1		486	752 + 28876	59.05
		2	16	32	600	426 + 29163	48.41
		4	8		435	423 + 20652	47.18
		8	4		428	421 + 19350	44.92
		16	2		541	423 + 24913	45.84
		4	16	64	650	255 + 27244	41.80
		8	8		528	257 + 21760	41.08
		16	4		570	258 + 23686	41.43
		8	16	128	710	181 + 30096	42.29
		16	8		627	183 + 26607	42.30

* 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.

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

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 $O(P \cdot A + n_0 B \log P)$ where A is the overhead constant and B the average cost per message.

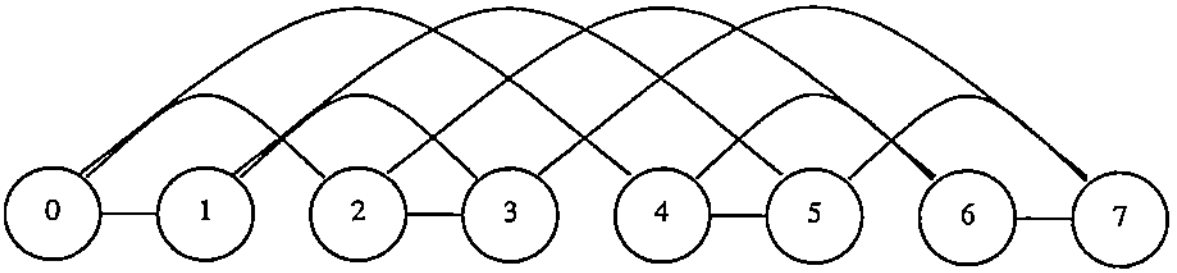
Following Stout [12], we have implemented a faster communication scheme for updating the solution vector on all processors where messages are grouped to produce bigger messages which in turn can be broadcast in less time. In this scheme, first the processors exchange vector parts with their lowest bit 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.

```
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 $\frac{n_0}{P} \sum_{i=0}^{\log P - 1} 2^i = n_0 \frac{P-1}{P}$. The above scheme turns out to be optimal, since the longest path among processors on a hypercube architecture is of order $\log P$.



Part of the vector the processors know

Processor	0	1	2	3	4	5	6	7
Step								
0	1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80
1	1-20	1-20	21-40	21-40	41-60	41-60	61-80	61-80
2	1-40	1-40	1-40	1-40	41-80	41-80	41-80	41-80
3	1-80	1-80	1-80	1-80	1-80	1-80	1-80	1-80

Figure 5.1. Exchange steps for a 3-dimensional cube and a vector of 80 components.

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.

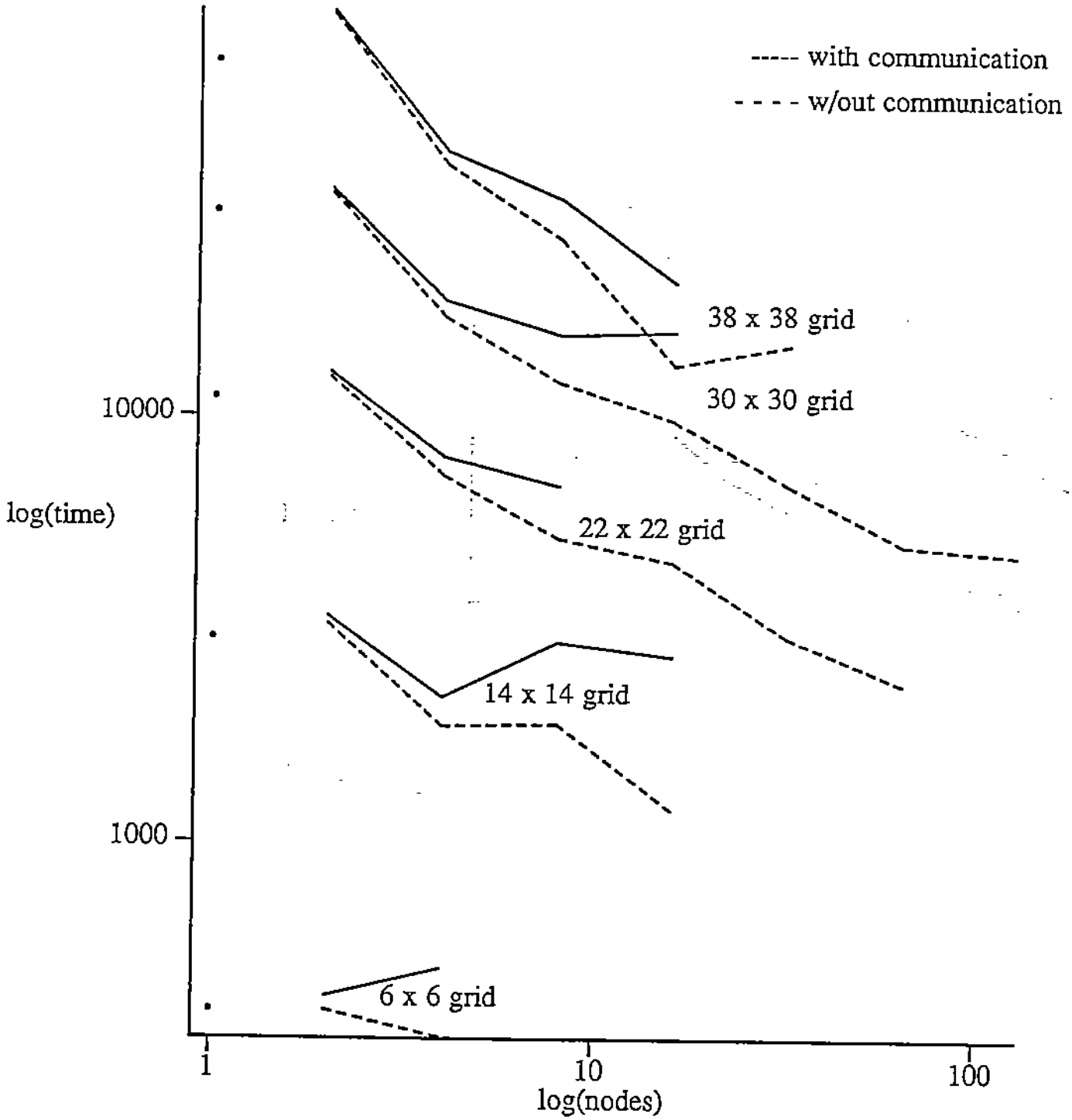


Figure 6.1. Timing of PQSCC algorithm in msecs on the NCUBE/7 with the acceleration constant ω equal to the computed optimal one. In this graph we plot the total time (solid line) and processing time (dotted line) versus the number of NCUBE processors (nodes) used for additional grids. The accelerated 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.

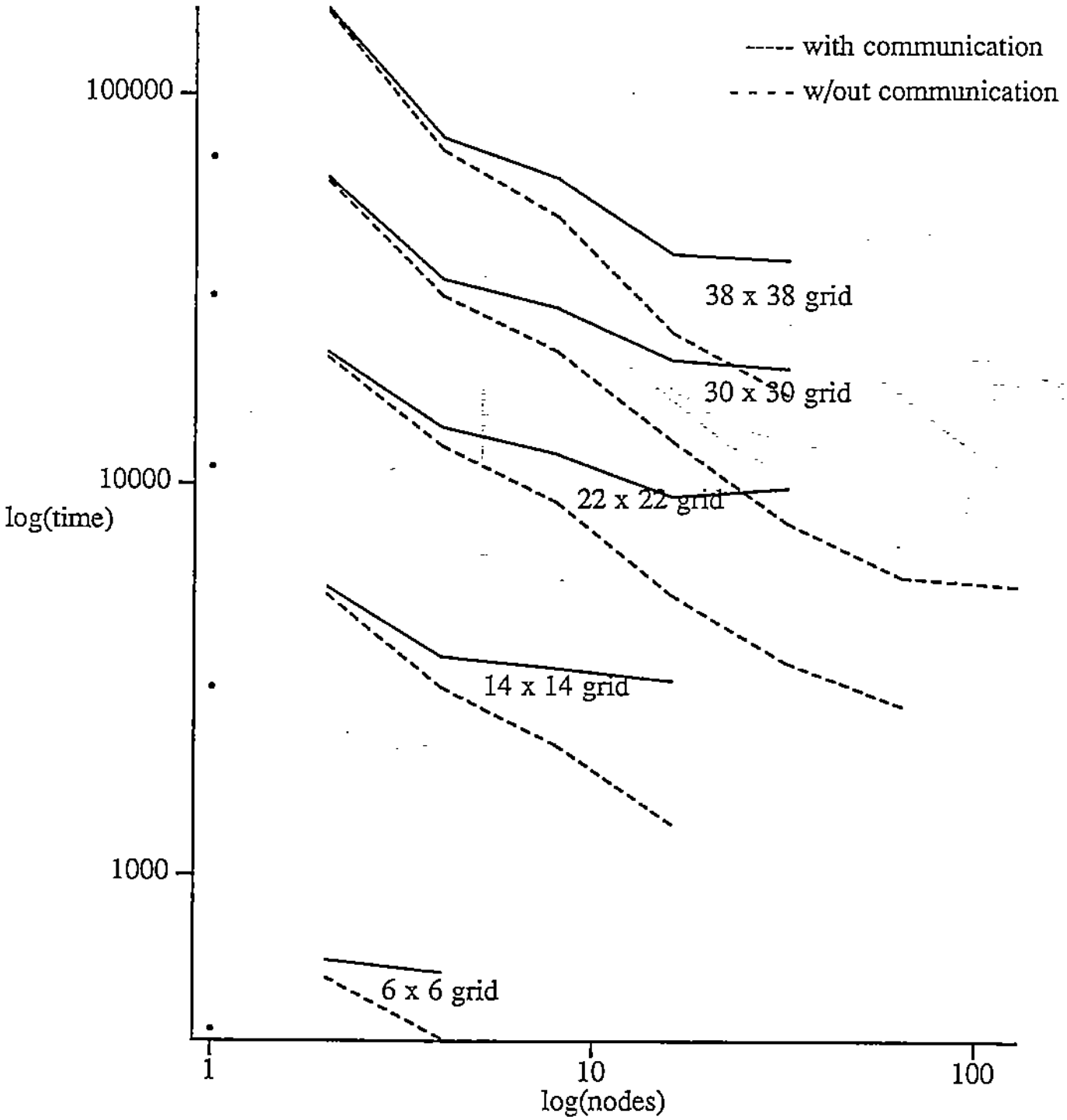


Figure 6.2. This graph is similar to the one in Figure 6.1 with $\omega = 1$ as the acceleration constant.

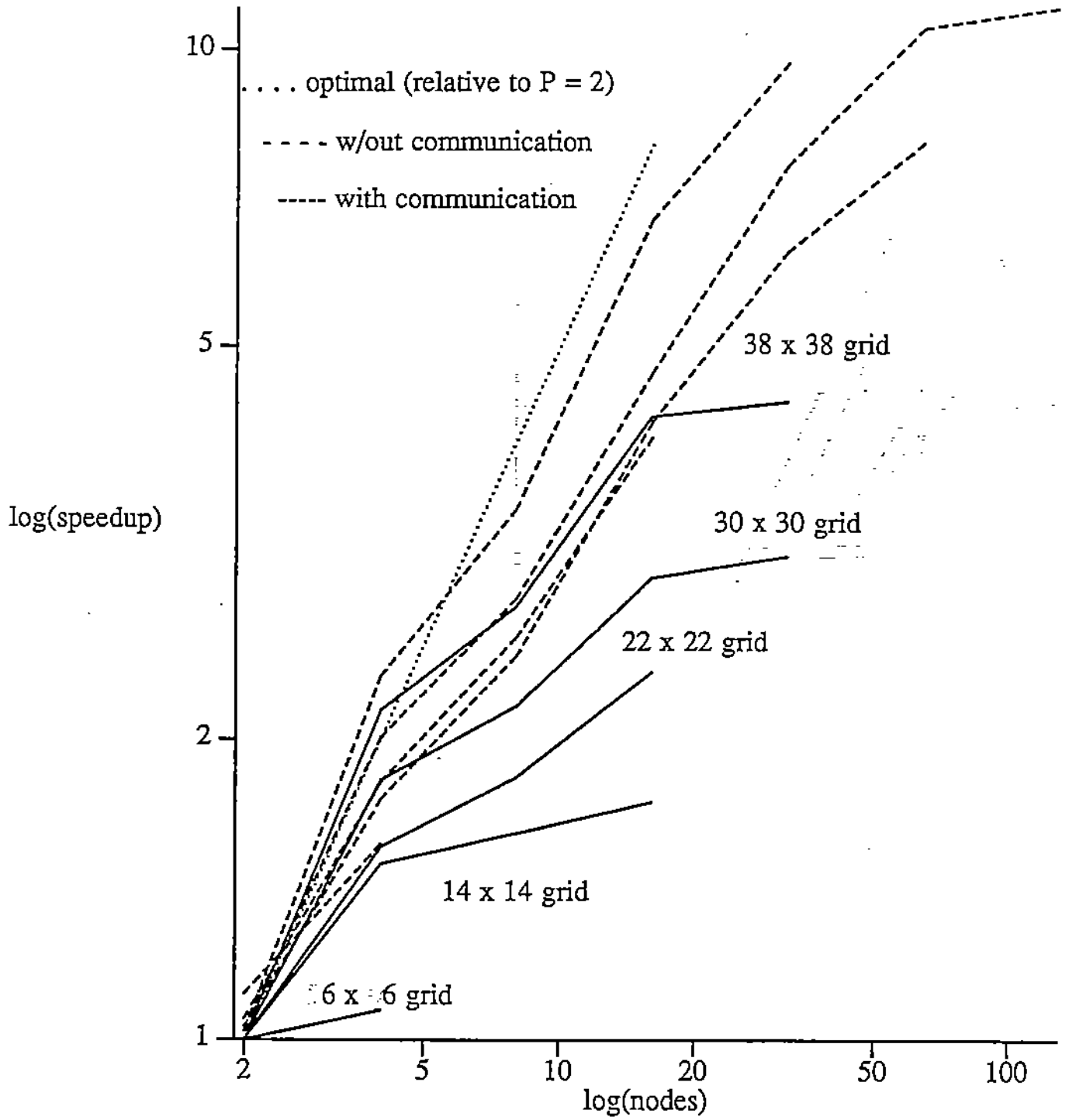


Figure 6.3 This graph corresponds to Figure 6.2 and plots the speedup vs number of processors. The speedup is determined with respect to the time of the PQSCC algorithm on two processors.

REFERENCES

[1] Peter E. Bjorstand and Olof B. Widlund, "Iterative methods for the solution of elliptic problems on regions partitioned into substructures", *SIAM J. Numer. Anal.*, 23, (1986), pp. 1097–1120.

[2] J.H. Bramble, J.E. Pasciak and A.H. Schatz, "The construction of preconditioners for elliptic problems by substructuring I", *Math. Comp.*, 47, (1986), pp. 103–134.

[3] J.H. Bramble, J.E. Pasciak and A.H. Schatz, "An iterative method for elliptic problems on regions partitioned into substructures", *Math. Comp.*, 46, (1986), pp. 361–369.

[4] Tony F. Chan, "Analysis of preconditioners for domain decomposition", *SIAM J. Numer. Anal.*, (1987), pp. 382–390.

[5] C.C. Christara, *Parallel Algorithms/Architectures for the Solution of Elliptic Partial Differential Equations*, Ph.D. Thesis, Purdue University, August 1988 (expected).

[6] Q.V. Dihn, R. Glowinski and J. Periaux, "Solving elliptic problems by domain decomposition methods with applications", *Elliptic Problem Solvers II*, Academic Press, (1984), pp. 395–426.

[7] M. Dryja and W. Proskurowski, "Iterative methods in subspaces for solving elliptic problems using domain decomposition", *Tech. Rpt. CRI-86-10*, University of Southern California, May 1986.

[8] M. Dryja, "A finite element – capacitance method for elliptic problems on regions partitioned into subregions", *Numer. Math.*, 44, (1984), pp. 153–168.

[9] E.N. Houstis, J.R. Rice, C.C. Christara and E.A. Vavalis, "Performance of scientific software", in: *Scientific Software*, (J.R. Rice, ed.), Springer Verlag, 1988.

[10] David E. Keys and William D. Gropp, "A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation", *SIAM J. Sci. Stat. Comput.*, 8, (1987), pp. s166–s202.

[11] Keith Miller, "Numerical analogs to the Schwartz alternating procedure", *Numer. Math.*, 7, (1965), pp. 91–103.

[12] Quentin F. Stout and Bruce Wagar, *Tech. Rpt.*, University of Michigan, Ann Arbor.

[13] John R. Rice and R. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer Verlag, 1984.

[14] Wei-Pai Tang, *Schwartz Splitting and Template Operators*, Ph.D. Thesis, Stanford University, June 1987.