Schur Complement Preconditioned Conjugate Gradient Methods for Spline Collocation Equations

Christina C. Christara
Department of Computer Sciences
University of Toronto
Toronto, CANADA, M5S 1A4

Abstract

We are interested in the efficient solution of linear second order Partial Differential Equation (PDE) problems on rectangular domains. The PDE discretisation scheme used is of Finite Element type and is based on quadratic splines and the collocation methodology. We integrate the Quadratic Spline Collocation (QSC) discretisation scheme with a Domain Decomposition (DD) technique. We develop DD motivated orderings of the QSC equations and unknowns and apply the Preconditioned Conjugate Gradient (PCG) method for the solution of the Schur Complement (SC) system. Our experiments show that the SC-PCG-QSC method in its sequential mode is very efficient compared to standard direct band solvers for the QSC equations. We have implemented the SC-PCG-QSC method on the iPSC/2 hypercube and present performance evaluation results for up to 32 processors configurations. We discuss a type of nearest neighbour communication scheme, in which the amount of data transfer per processor does not grow with the number of processors. The estimated efficiencies are at the order of 90%.

1. Introduction.

We study the parallel solution of the PDE problem

$$Lu \equiv aD_x^2u + bD_xD_yu + cD_y^2u + dD_xu + eD_yu + fu = g \quad (1.1)$$

$$\text{in } \Omega \equiv (ax,bx) \times (ay,by)$$

$$Bu \equiv \alpha u + \beta D_n u = g_0 \quad \text{on } \partial\Omega \equiv \text{boundary of } \Omega \quad (1.2)$$

The input functions in the PDE problem (1.1)-(1.2) are assumed to be functions of x and y in $C^1[\Omega]$, while D_n denotes the normal derivative of u on $\partial\Omega$.

In [Chri88b], [Chri90b] we have formulated and analysed a new method for the discretisation of the PDE problem (1.1)-(1.2). This discretisation method is of Finite Element type and is based on quadratic splines and the collocation methodology. The convergence of the Quadratic Spline Collocation (QSC) approximation to the solution of the above PDE problem is *optimal*, in the sense that it is of the same order as the interpolant in the quadratic spline space.

In this paper we study the integration of the QSC discretisation scheme with a *Domain Decomposition* (DD) technique, in which the domain Ω of problem definition is decomposed into non-overlapping subdomains. We develop appropriate orderings of the QSC equations and unknowns, and study the application of the Preconditioned Conjugate Gradient (PCG) method to the Schur Complement (SC) or capacitance system of QSC equations. Our experiments show that the number of PCG iterations required for convergence grows linearly with the square root of the number of equations of the SC system. We study the computational complexity of the SC-PCG-QSC method and show that it is an order less than standard direct band solvers for the QSC equations.

We discuss the implementation of the SC-PCG-QSC method on parallel machines. We focus on the hypercube architecture but most of the discussion can be straightforward extended to other MIMD architectures such as 2-dimensional grid of processors, as well as shared memory machines. Finally, we present the results from experiments that we have carried out on the iPSC/2 hypercube for up to 32 processors configurations. The fact that we have used quadratic splines does not limit the importance of our results, since the use of other degree splines gives rise to linear systems with similar properties to those of the quadratic spline equations. Discretisation methods other than spline collocation are known to give rise to similar structure linear systems.

In [Chri89], [Chri88a] we made the first attempt to apply DD methods for solving the PDE problem (1.1)-(1.2) and presented the results from the implementation of those on the NCUBE/7 and SEQUENT BALANCE 21000 machines. Iterative methods other than PCG were used for the solution of the SC system. In [Hous88b] *domain splitting* methods, in which the domain is decom-

posed into overlapping subdomains are integrated with cubic spline collocation and implemented on the NCUBE/7 hypercube. In [Chri90a] we discuss the application of Conjugate Gradient (CG) and PCG methods to the system of QSC equations.

Many researchers have studied the convergence and/or parallel implementation of PCG method applied to the linear systems arising from the discretisation of elliptic problems by other Finite Element Methods (FEMs), or to the SC systems arising from DD methods and appropriate reordering [Keye87], [Dryj84], [Bram86], [Bjor86], [Dryj86]. Others [Rodr86], [Tang87] experiment with domain splitting methods. The study of the solution of spline collocation equations is limited due to the fact that the development of optimal schemes for two-dimensional problems is very recent [Hous87], [Irod87], [Chri90b], and due to the lack of some nice properties, such as symmetry and positive definiteness, which are often standard properties for other FEM equations. This paper is the first successful study of the application of PCG methods to the capacitance system of spline collocation equations.

2. The Quadratic Spline Collocation method.

Spline collocation methods have been proven an efficient alternative for solving elliptic PDEs [Hous88a]. The general formulation of these methods for the discretisation of (1.1)-(1.2) was briefly presented in [Chri89] and we do not include it here. We include though for later reference the formulation of the QSC method for the discretisation of (1.1)-(1.2). We assume a uniform rectangular mesh $\Delta \equiv \{(x_i,y_j)$: $i = 0$ to n and $j = 0$ to $m\}$ in Ω, on which we define a tensor product of one-dimensional quadratic splines

$$S_{2,\Delta} \equiv S_{2,\Delta_x} \otimes S_{2,\Delta_y} \equiv \mathbb{P}_{2,\Delta} \cap C^1(\Omega)$$

with $\mathbb{P}_{2,\Delta}$, denoting the space of piecewise biquadratic polynomials with respect to Δ.

We define the set of collocation points T_Δ to be the set of midpoints of all subrectangles of Δ, including the midpoints of subintervals on the boundary and the four corners of Ω. A subset of T_Δ is the set T_∂ of collocation points that lie on the boundary of Ω. We determine the quadratic spline approximation $v \in S_{2,\Delta}$ to u in two steps by the following equations:

$$\textit{Step 1}: \quad \begin{aligned} Lv &= g \text{ on } T_\Delta - T_\partial \\ Bv &= g_0 \text{ on } T_\partial \end{aligned} \tag{2.1}$$

$$\textit{Step 2}: \quad \begin{aligned} Lu_\Delta &= g - P_L v \text{ on } T_\Delta - T_\partial \\ Bu_\Delta &= g_0 - P_B v \text{ on } T_\partial \end{aligned} \tag{2.2}$$

where P_L and P_B are appropriate perturbation operators, defined in [Chri90b]. The first step solution v is a second order approximation to u and u_Δ is a fourth order one.

The QSC equations (2.1) or (2.2) form a block tri-diagonal linear system, of $(n+2)\cdot(m+2)$ equations. If we assume that the ordering of the collocation points is bottom-up and then left-to-right (*natural* ordering) every block is of order $m+2$, the upper and lower bandwidth is $m+3$ and there are $n+2$ blocks on the diagonal. Figure 2.1 shows the pattern of non-zero entries in the QSC matrix.

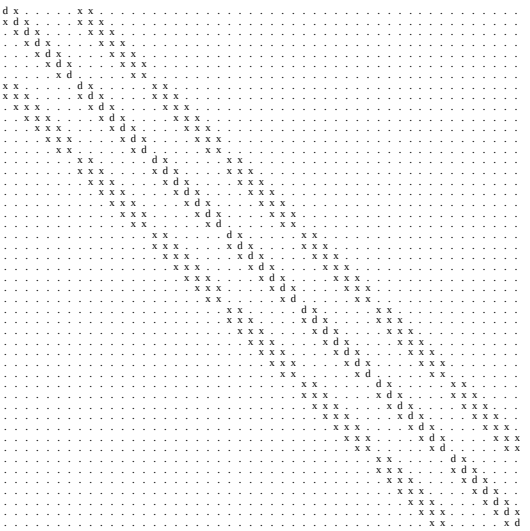

Figure 2.1. Structure of the matrix of QSC equations for $n = m = 5$. x denotes a non-zero off diagonal element, d a non-zero diagonal one, while all zero entries are represented by character ‘.’.

3. The Preconditioned Conjugate Gradient (PCG) method for solving linear systems.

In this section our aim is to recall some issues in the parallel implementation of the PCG method. For later reference we include here the steps of the PCG algorithm for solving a linear system $Ax = b$ with preconditioner M, as described in [Golu87]. The superscripts on vectors or scalars denote the iteration of the algorithm at which the vectors or scalars are computed.

PCG algorithm for $Ax = b$

1. $x^0 =$ initial guess
2. $r^0 = b - Ax^0$
   for $k = 1$, maxit
3. if $||r^{k-1}|| \le \varepsilon$ (or $||r^{k-1}|| \le ||r^0|| \varepsilon$) exit
   else
4. solve $Mz^{k-1} = r^{k-1}$
5. $\beta^k = z^{k-1^T} r^{k-1} / z^{k-2} r^{k-2}$
6. $p^k = z^{k-1} + \beta^k p^{k-1}$
7. $\alpha^k = z^{k-1^T} r^{k-1} / p^{k^T} A p^k$
8. $x^k = x^{k-1} + \alpha^k p^k$
9. $r^k = r^{k-1} - \alpha^k A p^k$
10. endif
    endfor

$x = x^{k-1}$

The computational requirements of every PCG iteration are discussed in detail in [Orte88]. From the above it is clear that the parallel implementation of the PCG method depends very much on the implementation of the linked triad operation (scalar-vector multiplication and vector addition), the inner product operation, the matrix-vector multiplication, the back-and-forward substitutions and the computation of the norm. There are numerous ways to implement the above operations on a parallel machine [Orte88]. They mainly reflect the assignment of the elements of the matrices *A* and *M* and the vectors *r*, *z*, *p* and *x* to the processors.

4. Domain Decomposition Methods.

This section is a general discussion about Domain Decomposition Methods (DDMs). DDMs have become increasingly attractive, since in addition to other advantages they are naturally parallelisable, thus serving the demand of scientists and engineers to solve large scale problems.

The first step in a DDM is the partition of the domain of problem definition into *non-overlapping* subdomains. With the partitioning of the domain into subdomains we indirectly achieve the subdivision of the geometric constructs of the domain, ie. points or elements of the grid, into subsets. Some of them lie in the (interior of the) subdomains and some lie on or constitute the interfaces between subdomains. For the case of rectangular domains the most common partitions are the *stripwise* and the *boxwise* decompositions [Keye87]. Assuming that a PDE discretisation method has been chosen, the next step in a DDM is the ordering of the equations and unknowns of the linear system arising from the discretisation of the PDE problem. This ordering is motivated by the specific partitioning of the domain applied as above in the following sense: The data points or data elements of the domain are ordered, with the interface points/elements coming first, followed by the points/elements of each subdomain in order. Then the equations and unknowns of the system are ordered similarly, following the (often if not always) one-to-one mapping of the data points or elements to the equations and unknowns.

The result of the domain decomposition ordering of the equations/unknowns is the so called *arrowhead matrix* [Orte88]. That is, the discretised PDE takes the form of the system (4.1)

$$\begin{bmatrix} A_{00} & B_1 & B_2 & . & . & B_P \\ C_1 & D_1 & & & & \\ C_2 & & D_2 & & & \\ . & & & . & & \\ . & & & & . & \\ C_P & & & & & D_P \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_{11} \\ x_{12} \\ . \\ . \\ x_{1P} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_{11} \\ b_{12} \\ . \\ . \\ b_{1P} \end{bmatrix} \qquad (4.1)$$

where the first block of equations/unknowns ($A_{00}, B_1, B_2, ..., B_P$ and x_0) corresponds to the interface points/elements and each subsequent block (C_i, D_i and x_{1i}, $i = 1,...,P$) corresponds to a subdomain's points/elements. System (4.1) can be written as a 2×2 block system as in (4.2).

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \qquad (4.2)$$

where A_{00} represents the relation between interface equations/unknowns, A_{11} the relation between subdomains' equations/unknowns and A_{01}, A_{10} represent the interaction between subdomains and interfaces.

By applying block Gauss elimination to the above partitioned system (4.2) we eliminate the subdomains' unknowns and obtain the so called *capacitance* or *Schur Complement* (SC) system

$$Cx_0 = w \qquad (4.3)$$

where $C = A_{00} - A_{01}A_{11}^{-1}A_{10}$ and $w = b_0 - A_{01}A_{11}^{-1}b_1$. Once (4.3) is solved for x_0 one computes x_1 by solving

$$A_{11}x_1 = b_1 - A_{10}x_{0.} \qquad (4.4)$$

There is a variety of methods to solve (4.2) and (4.3). Very often PCG iterations are used. Following the terminology of [Keye87] we refer to methods that depart from (4.2) as *Partitioned Matrix Methods* (PMMs) and those that depart from (4.3) as *Schur Complement Methods* (SCMs). In the rest of the paper we formulate, implement and evaluate SCMs and PMMs for the QSC equations.

5. Domain Decomposition Methods for QSC equations.

5.1. The Domain Decomposition ordering of the QSC points.

We first partition the domain Ω into $p_n \times p_m$ subdomains (boxes). The case of stripwise decomposition can be considered as a special case of $p_n \times 1$ or $1 \times p_m$ boxes. With this partition we obtain $(p_n-1)\cdot p_m$ and $(p_m-1)\cdot p_n$ vertical and horizontal interfaces respectively including the $(p_n-1)\cdot(p_m-1)$ cross-points. For simplicity we assume that p_n divides $n+3$ and p_m divides $m+3$, so that every subdomain has exactly the same distribution of collocation points (but not necessarily the same area). We then number the collocation points as follows: We number the interface points first (their relative ordering will be discussed in Section 5.2) and then the subdomains' points according to a *natural* ordering (left to right then bottom-up or vice-versa) for each subdomain in the natural order. The ordering of the collocation equations/unknowns follows the numbering of the collocation points.

With the above ordering of the collocation equations/unknowns the matrix of QSC equations takes the form of the arrowhead matrix of (4.1), where every block D_i, $i = 1,...,P \equiv p_n \cdot p_m$ is an $(\frac{n+3}{p_n}-1)\cdot(\frac{m+3}{p_m}-1)$ order banded matrix with bandwidth $\frac{n+3}{p_n}$ and A_{00}, A_{01} and A_{10} are sparse.

5.2. The ordering of the interface QSC points.

The ordering of the interface QSC points has been irrelevant so far, since any ordering would give us the arrowhead matrix (4.1). Since our aim is the application of the PCG method to (4.2) or (4.3) and its efficient implementation on MIMD architectures, we try to develop an ordering so that appropriate preconditioners can be constructed and best load balance and minimum communication can be achieved. In this section we study how the ordering of the interface QSC points affects the structure of the sparse matrices A_{00}, A_{01} and A_{10}.

5.2.1. The Best Load Balance ordering.

We first experiment with the following ordering of the interface QSC points: We number the horizontal interfaces in a natural order and then similarly the vertical ones, which are assumed to include the cross-points. The points on each interface in order are numbered in a natural order. If we order the QSC equations/unknowns following the above numbering of the interface and subdomains' points, the matrix A_{00} will be "mostly" tridiagonal, that is, it will have as many entries on the sub-, main- and super-diagonals as possible. These entries can be used for constructing preconditioners for the capacitance matrix C, or for formulating iterative methods other than PCG for solving (4.3). The disadvantage of this ordering is that it does not result in a particular structure for the B_i and C_i matrices of (4.1), which we could take advantage of, in the implementation of an iterative method for solving (4.3). Thus, the implementation of the PCG method applied to the capacitance system of the QSC equations with this ordering requires global communication. The advantage is that any iterative method for solving (4.3) can be formulated so that *best load balancing* is achieved, in the sense that the difference between the number of equations/unknowns, that two processors deal with is not more than one. Iterative methods for solving the capacitance system resulting from the above ordering have been formulated, studied and implemented. The results are presented in [Chri88], [Chri89], so we will not include them here. We will focus on a different ordering which minimises communication.

5.2.2. The Second Nearest Neighbour ordering.

We assume that the east and north interface of each subdomain are attached to it, and number the interface points of each subdomain in order as follows: Number the east (vertical line) interface points bottom-up, then the north-east cross-point, then the north interface points right to left. Figure 5.1 shows the SNN numbering of the QSC points for a 6×6 grid and a 2×2 domain decomposition. Figure 5.2 shows the structure of the QSC equations for the same grid and domain decomposition. With this ordering we still achieve to have as many tridiagonal entries in A_{00} as possible. In addition, the tridiagonal part of A_{00} and matrices B_i and C_i of (4.1) possess some *locality* property, which will be discussed in Section 6.3, and which explains the title of this subsection. We will refer to this ordering as Second Nearest Neighbour (SNN) ordering.

Figure 5.1. The SSN numbering of the QSC points for a 6×6 grid and a 2×2 domain decomposition. The interface points are denoted by circles, while the points interior to subdomains are denoted by squares.

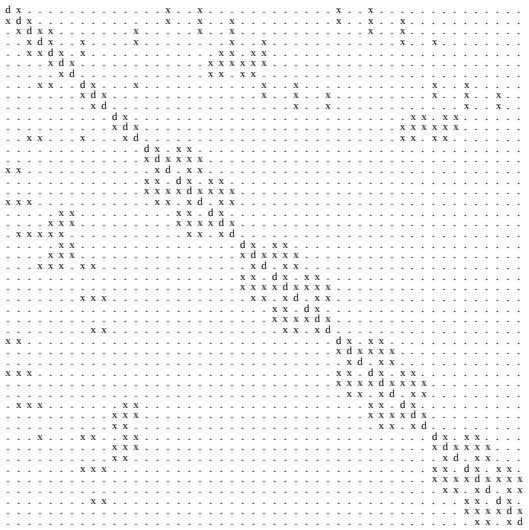

Figure 5.2. Structure of the matrix of QSC equations for a 6×6 grid ($n = m = 5$) and a 2×2 domain decomposition according to the SNN ordering. The notation of Figure 2.1 is used.

5.3. The Schur Complement PCG method for QSC equations.

5.3.1. Preconditioner.

As the above discussion on the SNN ordering implies our goal is the application of the PCG method to the capacitance system arising from the above ordering of the QSC equations/unknowns. It is quite common for matrices of the structure of A_{00}, which is "mostly" tridiagonal as explained above, to use the tridiagonal part T of it as preconditioner. More specifically, each row of A_{00} can have at most 5 non-zero entries, and at most 3 of them might not belong to the respective row of T. We should note that A_{00} is a "good" approximation of the capacitance matrix C, in the sense that the entries of A_{00} are much more dominant in absolute value than the entries of C coming from the $A_{01}A_{11}^{-1}A_{10}$ part of it. In the case of QSC capacitance system and with the SNN ordering the tridiagonal part T of A_{00} is also block diagonal, with one block corresponding to each interface. This latter property comes from the fact that interface points are numbered "continuously" without "jumps" on the cross-points. Other SNN orderings that do not number the points of each interface continuously may have more than one block corresponding to each interface, thus reducing the number of non-zero entries of T, and leaving less information on the preconditioner. In Figure 5.2 the first block of T has 7 rows, formed by the equations corresponding to points 1-7 of Figure 5.1, the second block of T has 3 rows, formed by the equations corresponding to points 8-10 of Figure 5.1, and the third block of T has 3 rows, formed by the equations corresponding to points 11-13 of Figure 5.1.

5.3.2. The method.

At this point we briefly describe the Schur Complement Preconditioned Conjugate Gradient method for QSC equations (SC-PCG-QSC). It consists of the following steps:

1. Solve $A_{11}\hat{x}_1 = b_1$,
2. Compute $w = b_0 - A_{01}\hat{x}_1$,
3. Solve $Cx_0 = w$ with PCG and preconditioner T,
4. Solve $A_{11}x_1 = b_1 - A_{10}x_0$.

We note that step 1 above involves the factorisation and forward-and-back substitutions for the banded matrices D_i, $i = 1,...,P$ of (4.1). Since during the application of the PCG algorithm to C, the matrix C itself does not get involved except for computing the product of C with a given vector (lines 7 and 9 of the PCG algorithm), the explicit computation of the entries of C is avoided by carrying out forward-and-back substitutions for the factored matrices D_i, $i = 1,...,P$ in each iteration of the PCG algorithm.

5.3.3. Convergence of the PCG method on the capacitance system.

Our experiments show that the number of PCG iterations applied to the capacitance system C with preconditioner T, that are required to satisfy a stopping criterion as in line 3 of the PCG algorithm grows linearly with the square root of the order of C, ie. the number of interface equations/unknowns. Table (5.1) shows the number of iterations *nit* required for the SC-PCG-QSC method when applied to the problem

$$u_{xx} + u_{yy} = f \quad \text{in } \Omega = (0,1)\times(0,1) \tag{5.1a}$$

$$u = g \quad \text{in } \partial\Omega, \tag{5.1b}$$

where f and g are chosen so that the solution to the problem is $u(x,y) = x^{13/2}(x-1)y^{13/2}(y-1)$. The initial guess for x_0 that was used for step 1 of the QSC method was the zero vector, while in step 2 we use the already computed solution vector x_0 from step 1. The relative Euclidean norm of the residual was used for the stopping criterion in line 3 of the PCG algorithm. The desired precision ε was set to 10^{-5}. The number of iterations for domain decompositions $p_n \times p_m$ was equal to that of decompositions $p_m \times p_n$ for this PDE operator (5.1) so we do not include both entries in the table. The last column of Table 5.1 shows the time in seconds for the 4 steps of SC-PCG-QSC. The computation was carried out on (one processor of) a SUN-4/60 workstation in double precision.

In order to study the growth of the number of iterations *nit* with the parameters of the problem in more detail, we first take a look at the formula that gives us the number of interface points. Let n_0 be the number of

interface points.

$$n_0 = (n+2)(p_m-1) + (m+2)(p_n-1) - (p_n-1)(p_m-1) \quad (5.2a)$$

This means that

$$n_0 = O(n(p_n+p_m)) = O(n(p_n+\frac{P}{p_n})) \quad (5.2b)$$

assuming for simplicity that $n = m$ and taking in account that $P = p_n \cdot p_m$. Relation (5.2b) shows that there are 3 parameters in n_0, namely the grid size n, the total number of processors P and the domain decomposition, ie. 1 more independent variable p_n. For constant grid size and total number of processors, n_0 is minimised when $p_n = p_m = \sqrt{P}$. This agrees perfectly with the experimental results shown in Table 5.1 and with the results of other researchers [Keye87] on SCMs for discretisation methods other than spline collocation.

Table 5.1. Number of iterations required for the convergence of the PCG method applied to the Schur Complement system (4.3) of the QSC equations (2.1)-(2.2) corresponding to Problem (5.1) for several grid sizes and domain decompositions.

grid size	number of subdomains	domain decomp.		rows in C	number of iterations		solution time
$n+1$	P	p_n	p_m	n_0	nit		$t(n,P,p_n)$
					step 1	2	step 1
46	2	2	1	47	16	11	11.3
	4	2	2	93	17	12	11.3
	4	4	1	141	20	13	6.9
	8	4	2	185	23	16	7.4
	8	8	1	329	26	17	6.1
	16	4	4	273	24	17	7.7
	16	8	2	369	29	21	6.7
	16	16	1	705	36	22	7.2
	32	8	4	449	34	21	8.0
	32	16	2	737	40	26	8.2
	64	8	8	609	33	23	
	64	16	4	801	43	27	
	128	16	8	929	46	28	
94	2	2	1	95	23	13	171.8
	4	2	2	189	24	15	172.8
	4	4	1	285	29	15	86.7
	8	4	2	377	34	19	96.0
	8	8	1	665	37	20	57.9
	16	4	4	561	35	21	95.7
	16	8	2	753	42	23	63.7
	16	16	1	1425	47	28	51.6
	32	8	4	929	48	26	71.2
	32	16	2	1505	58	31	63.2
	32	32	1	2945	66	41	52.0
	64	8	8	1281	48	28	
	64	16	4	1665	61	33	
	128	16	8	1985	61	36	

We next consider a constant grid size and vary the total number of processors P so that $p_n = P$ (and $p_m = 1$). In this case the number of iterations grows linearly with $\sqrt{P}$ and this is shown graphically in Figure 5.3(a). Figure 5.3(b) shows the case of having constant grid size and varying P so that $p_n = p_m = \sqrt{P}$, in which case the number of iterations grows linearly with the fourth root of the total number of processors P. Finally in Figure 5.4 we fix both P and p_n and vary n. These results compare well with those in [Chri90a], where the CG method is applied directly to the original block tridiagonal matrix of QSC equations with the natural ordering (CG-QSC method) and where we show that the number of iterations in that case grows linearly with n.

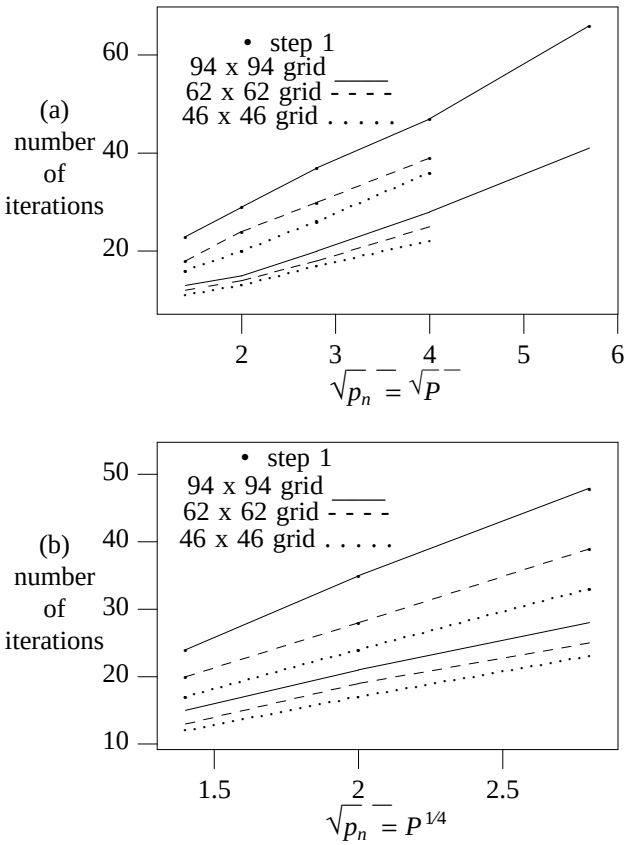

Figure 5.3. Plot of the number of iterations required for the convergence of SC-PCG-QSC method versus the square root of the number of processors in one direction for several grid sizes and for the 2 steps of the QSC method. Among the various domain decompositions we choose the one with $p_m = 1$, so that $P = p_n$ for (a), and the one with $p_n = p_m = \sqrt{P}$ for (b).

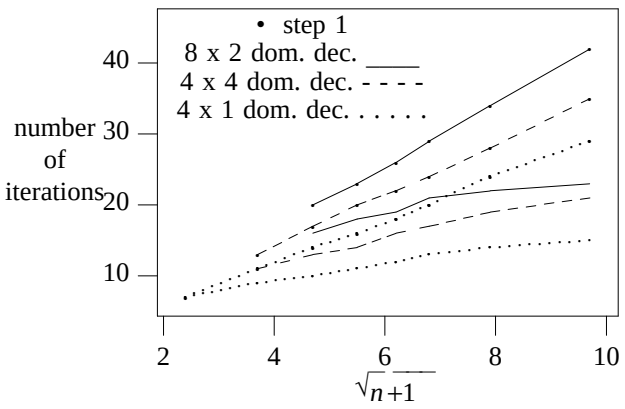

Figure 5.4. Plot of the number of iterations required for the convergence of SC-PCG-QSC method versus the square root of the size of the grid in one direction for several domain decompositions and for the 2 steps of the QSC method.

It is interesting to note that we have successfully applied the SC-PCG-QSC method to PDE problems with operators other than the Laplace operator and the asymptotic convergence of the PCG iterations was not very different from the one for Problem 5.1. We also wish to note that the CG method without preconditioner applied to the SC system of QSC equations did not converge or converged very slowly.

5.3.4. Performance of the SC-PCG-QSC method.

In this section we study the performance of the SC-PCG-QSC method as compared with standard banded Gauss elimination without pivoting (Band-LU), and with standard CG method (CG-QSC). For this reason we study the computational complexity of SC-PCG-QSC as defined in Section 5.3.2. We consider steps 1, 2 and 4 of SC-PCG-QSC as the *direct part* and step 3 as the *iterative part* of the method. It turns out that the most time consuming operation in the direct part of SC-PCG-QSC is the factorisation of the banded blocks D_i, $i = 1,..,P$ in step 1 of SC-PCG-QSC and in the iterative part the forward-and-back substitutions in each PCG iteration of Step 3. This shows that the computational complexity of the SC-PCG-QSC method is $O(\frac{n^4}{p_n^2} + nit\cdot\frac{n^3}{p_n})$, where we have again assumed that $n = m$ and recalled that the bandwidth of the blocks D_i is $O(\frac{n}{p_n})$. According to our results in Section 5.3.3 for the asymptotic behaviour of *nit* the computational complexity of SC-PCG-QSC method is

$$O(\frac{n^4}{p_n^2} + n^{7/2}(\frac{1}{p_n}+\frac{P}{p_n^3})^{1/2}) \qquad (5.3a)$$

or

$$O(\frac{n^4}{p_n^2} + n^{7/2}(\frac{1}{p_n}+\frac{p_m}{p_n^2})^{1/2}) \qquad (5.3b)$$

which, for fixed n and P, is minimised when $p_m = 1$ and $p_n = P$, ie. for stripwise decompositions. For the latter decompositions the computational complexity of SC-PCG-QSC becomes $O(\frac{n^4}{P^2} + \frac{n^{7/2}}{P^{1/2}})$. This also shows that for small P ($P < n^{1/3}$) the direct part of SC-PCG-QSC dominates in time, while for large P ($P > n^{1/3}$) the iterative part dominates. In the extreme case that P grows linearly with n, ($P = \frac{n+3}{2}$ or $P = \frac{n+3}{3}$ etc.) the time complexity of SC-PCG-QSC is $O(n^3)$ which is equivalent to the time complexity of the CG algorithm applied to the QSC equations (2.1) [Chri90a], and an order less than Band-LU for the same equations. Relations (5.3) also show that decompositions of $O(n)\times O(n)$ subdomains are equivalent to those of $O(n)\times 1$ subdomains, ie. we do not benefit from increasing the number of subdomains in the direction that does not affect the bandwidth of the D_i blocks. Note that all these hold for step 1 of the QSC equations, since in step 2 all methods are $O(n^3)$.

In Figure 5.5 we plot the solution time for the Band-LU, CG-QSC and SC-PCG-QSC methods for several grid sizes. Among the several domain decompositions for SC-PCG-QSC we have chosen the one that requires the least time, usually the $P\times 1$ decomposition with P growing linearly with n. The slope of the graphs corresponding to the SC-PCG-QSC and CG-QSC times is clearly lower than that of Band-LU.

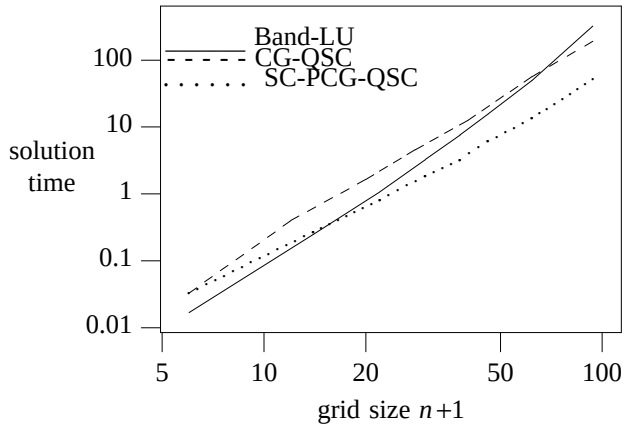

Figure 5.5. Plot of the time in seconds for the solution of step 1 of the QSC equations with the Band-LU, CG-QSC and SC-PCG-QSC methods versus the grid size in log-log scale. The computation was carried out on (one processor of) a SUN-4/60 workstation in double precision.

5.4. The Partitioned Matrix PCG method for QSC equations.

In this section we briefly formulate a Partitioned Matrix Preconditioned Conjugate Gradient method for QSC equations (PM-PCG-QSC). We intend to describe the method and discuss its performance in more detail, in a later publication.

Having adopted the background from the previous subsections on the ordering of the QSC equations/unknowns according to some domain decomposition, what remains for completing the formulation of a PMM for QSC equations is the choice of appropriate preconditioner. Our first try was to apply the PCG method to the reordered with the SNN ordering system (4.1) (or (4.2)) and use as preconditioner the block diagonal matrix $diag(T, D_1, D_2, \ldots, D_P)$. It turns out that this method requires slightly more iterations to converge, than the PCG method with preconditioner T on the SC system (4.3) of the QSC equations, but the slope of the number of iterations versus the number of interface equations is about the same for both the PM and the SC methods. More detailed study of the convergence of the PM-PCG-QSC method and its implementation on MIMD architectures will appear in the near future.

6. Implementation of the SC-PCG-QSC method on hypercube architectures.

In this section we discuss in more detail how the computation involved in the SC-PCG-QSC method is mapped on hypercube architectures. Although we limit this discussion in a specific MIMD architecture, most of the ideas presented are straightforward implemented on other type of local memory machines as well as shared memory ones.

6.1. Distribution of the data to processors.

The first thing in the implementation of an algorithm, in which certain parallelism is identified, on a specific local memory machine, is the distribution the data in the local memory of the processors, so that communication is "minimised" and as little as possible data is duplicated. In the case of SC-PCG-QSC method the distribution of data is motivated by the domain decomposition, which is reflected in the structure of the partitioned matrix (4.1) of QSC equations, as well as by the parallel implementation of the individual steps of the PCG algorithm as described in Section 3. Although this distribution of data holds only for local memory machines in the case of shared memory machines this distribution reflects the way the processors are going to address the shared memory.

The matrix A of QSC equations is stored in a distributed among the processors way as follows: Every processor is assigned one of the subdomains, including its (north and east) interface. The assignment of the blocks D_i, $i = 1,...,P$ to the processors follows the assignment of the subdomains. At this point we assume that we partition the domain in exactly as many subdomains as the number of available processors. We do not intend to argue that this is the only possible, or the best assignment of the blocks/subdomains to the processors, since in Section 7.2 we present performance results from other assignments too, for example assigning more than one block to a processor and we believe that under certain conditions (involving the number of available processors, the interconnection network, the limit on the amount of data transfer between processors, and the available memory per processor) other assignments might be more effective than the one-to-one mapping of subdomains to processors. The D_i blocks are stored as banded matrices in the local memory of the processors. The assignment of the interface equations to the processors also follows the assignment of the subdomains. This means that every processor is assigned the rows of A_{00} and A_{01} corresponding to the equations for the interface points of its own subdomain. Finally every processor is assigned the rows of A_{10} corresponding to the respective D_i block. The sparse matrices A_{00}, A_{01} and A_{10} are stored by rows, in a sparse matrix storage scheme, that allows efficient access for the multiplication of these matrices by vectors.

There are several vectors involved in the implementation of the SC-PCG-QSC method. Their components are also distributed among the processors in a similar way as the interface and interior to subdomains rows/equations of A. More specifically for the unknown vector x a processor has storage for the components corresponding to its own subdomain, plus some neighbour components, which will be discussed in detail in Section 6.3. The right-hand-side of the original system and of the SC system as well as vectors r and z of the PCG algorithm are totally local to the processors. As for the direction vector p this requires some "neighbour" storage as well.

Finally, the rows of the preconditioner T corresponding to the processor's interface equations are stored in the local memory of a processor in a tridiagonal matrix form.

6.2. Parallel discretisation of the PDE problem.

The discretisation process of a PDE problem with the collocation methodology is by definition pointwise, so it is totally asynchronous, assuming a distribution of the collocation points to the processors. In our case of the domain decomposition driven distribution of the collocation points, and the SNN numbering of them, the parallel generation of the matrix A can be viewed as a *subdomain* or *box* collocation method. Every processor generates the rows of A and the components of the right-hand-side it is assigned to, that is, the equations corresponding to one

subdomain including the interface, with no need to communicate with any other processor.

6.3. Computing the product of C by a vector.

As discussed in Section 5.3.2 the application of PCG method to C does not require the explicit evaluation of the entries of C, which would be time (and memory) consuming, since C involves the inverse of A_{11} (and C is dense). In order to compute the product q of C by a vector v we follow the steps below:

1. Compute $t = A_{10} \cdot v$.
2. Solve $A_{11}\hat{t} = t$.
3. Compute $t = A_{01} \cdot \hat{t}$.
4. Compute $s = A_{00} \cdot v$.
5. Set $q = s + t$.

We now discuss in detail the implementation of these steps on hypercube architectures. We assume that the processors of the hypercube are arranged in a two-dimensional grid, with an one-to-one mapping of subdomains to processors. This can be accomplished using the well known gray code. With the above layout of the processors/subdomains and the SNN ordering of the interface QSC equations/unknowns the multiplication of A_{10} by v reflects the interaction between the neighbour interface components of v with the interior components of a processor/subdomain. More specifically, when the central processor of Figure 6.1 computes $A_{10} \cdot v$, it has to know the components of v belonging to its south, west and south-west neighbours, which correspond to the east interface of its west neighbour (including the cross-point), the cross-point of its south-west neighbour and the north interface of its south neighbour (including the cross-point). Figure 6.1 shows all the points of the neighbours of a processor that interact with the processor during the multiplication of C by v. The points denoted by a circle represent the interaction of the interior of the north, east and north-east neighbours with the interface of the central subdomain (multiplication with A_{01}). Similarly the boxes represent the interaction of the south, west and south-west neighbour interfaces with the interior points of the central subdomain (multiplication with A_{10}). Finally, the ellipses represent the effect of the neighbour interfaces to the interface of the central subdomain (multiplication with A_{00}). It is also clear that there is no interaction between interior points of neighbour (or non-neighbour) subdomains.

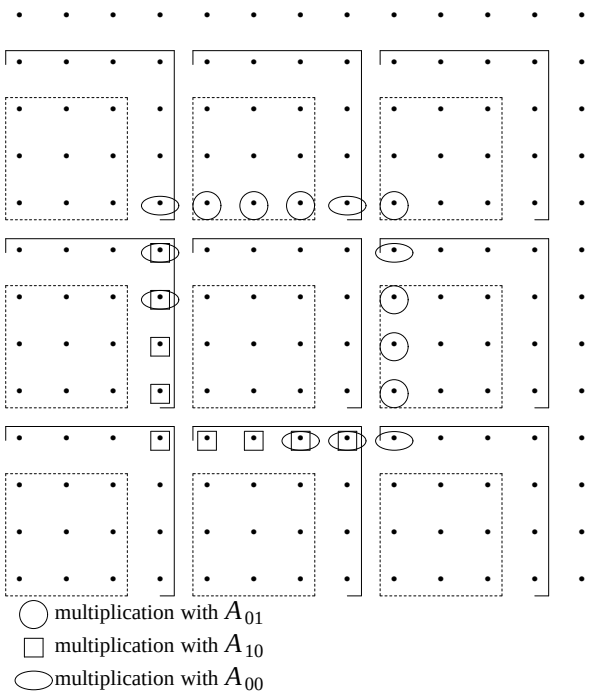

Figure 6.1. The interaction of neighbour points with a subdomain. The interior to subdomains points are inside the dashed lines, while the interface points inside the solid lines.

It is clear now why we call the ordering of Section 5.2.2 *Second Nearest Neighbour* ordering. At this point we want to note, that spline collocation methods in contrast with Finite Difference methods, do not use a stencil of values of the unknown function u of (1.1)-(1.2) in order to approximate the derivatives of u appearing in the operator (1.1)-(1.2), but they do use a stencil of points, which shows the interaction between the unknown degrees of freedom in the Finite Element representation of the spline approximation to u. In the case of quadratic splines (as well as for cubic splines) this is a *9-point-star* stencil.

Although the iPSC/2 hypercube, on which we implemented the SC-PCG-QSC method allows sending and receiving data to and from non-direct neighbours there is a way to implement the data transfer required for computing $C \cdot v$ using only direct links and with no more calls to the send and receive routines than in the case of a 5-point-star stencil, where there is interaction between direct neighbours only. Below is the code in a very high level language (excluding arithmetic operations) that a processor executes in order to compute $C \cdot v$, assuming it knows its own components of v only (but has storage for the neighbour components too).

Data Transfer Algorithm for 9-point-star stencil.

1. Send north interface components to north neighbour.
2. Receive north interface components from south neighbour.
3. Append north-east interface component of south neighbour at the end of my east interface components.
4. Send my east interface components (+1) to east neighbour.
5. Receive east interface components (+1) from west neighbour.
6. Copy last received component to the right position.
7. Append my south-east interface component and south-east interface component of west neighbour at the end of my south interior components.
8. Send my south interior components (+2) to south neighbour.
9. Receive south interior components (+2) from north neighbour.
10. Copy last 2 received components to the right positions.
11. Append my north-west interface component, south-west interior component of north neighbour and north-west interface component of south neighbour at the end of my west interior components.
12. Send my west interior components (+3) to west neighbour.
13. Receive west interior components (+3) from east neighbour.
14. Copy last 3 received components to the right position.

The computation of $A_{10} \cdot v$ can start as soon as line 6 of the above algorithm has been completed. Thus it can be overlapped with subsequent data transfers. The only difference of the above algorithm from the one for a 5-point star stencil is that data transfers cannot overlap. This makes the second nearest neighbour communication a *four-distinct-phases* nearest neighbour communication, in contrast with the one for the 5-point star stencil which is a *four-(possibly overlapping)-phases* nearest neighbour communication.

The amount of data transfer for each send/receive is about the same as the number of points of a subdomain in one direction (+1 or +2), which in turn is bounded by $n+2$, where $n+1$ is the grid size in one direction.

The above described data transfer algorithm would also work for other degree spline collocation methods, as well as for other Finite Element methods. For example, in the case of quadratic spline Galerkin, in which the linear system that arises is block 5-diagonal with the natural ordering, the SNN ordering would still give us a second nearest neighbour interaction between subdomains, assuming that the interior of each subdomain has at least 4 points/elements.

6.4. Preconditioning.

Since T is block diagonal, with one block corresponding to each subdomain, every processor can work independently for the back-and-forward substitutions of the block of T it is assigned to. So preconditioning with T does not increase the communication overhead.

6.5. Computation of inner product of vectors and norms.

For the inner products in lines 5 and 7 of the PCG algorithm the well known fan-in technique is used. More specifically, we use *global fan-in* so that the final result resides in all processors, instead of a fan-in in one processor and a fan-out broadcast of the final result to all other processors. The parallel computation of the norm of the vector depends on the norm used. For the infinity norm a global fan-in comparison scheme is used, while for the Euclidean norm a global fan-in summation.

7. Performance results.

In this section we discuss the performance of the SC-PCG-QSC method on various processor configurations of the iPSC/2 hypercube. We refer to the basic computational constructs of the SC-PCG-QSC method as *solution process* and *per iteration process.* The per iteration process includes the computation of one PCG iteration on the capacitance system, while the solution process includes the direct and iterative part of the method. The latter is also referred to as *iteration process*. We also use the term *discretisation process* referring to the computation required in order to generate the matrix A and the right-hand-side of the system (4.2).

7.1. Speedup and efficiency.

For any parallel algorithm that is implemented on a real parallel machine, it is common to measure its performance, by estimating the speedup of the algorithm as the number of processors increases. The speedup of an algorithm is in general defined as the ratio between the time for the algorithm to run on one processor over the time for the same algorithm to run on more than one processors. In the case though of SC-PCG-QSC method, with the one-to-one mapping of the subdomains to processors and the ordering of the equations/unknowns according to the domain decomposition this is not possible, strictly speaking. This happens because the algorithm/problem itself is adapted according to the domain decomposition and its serial time complexity changes as we increase the number of subdomains/processors.

We estimate the "fixed speedup" for the solution and per iteration processes of the algorithm by computing the ratio of the time for the algorithm to run on 2 processors with a 2×1 domain decomposition for some fixed grid size, over the time for the algorithm to run on more than 2 processors for the same grid size and multiplying the result by 2. On the other hand, for the discretisation process we measure the speedup relative to the discretisation for the natural ordering on one processor. Figures 7.1 and 7.2 plot the speedup (measured as explained above) of the discretisation, solution and per iteration processes for 2 grid sizes 94×94 and 62×62 respectively.

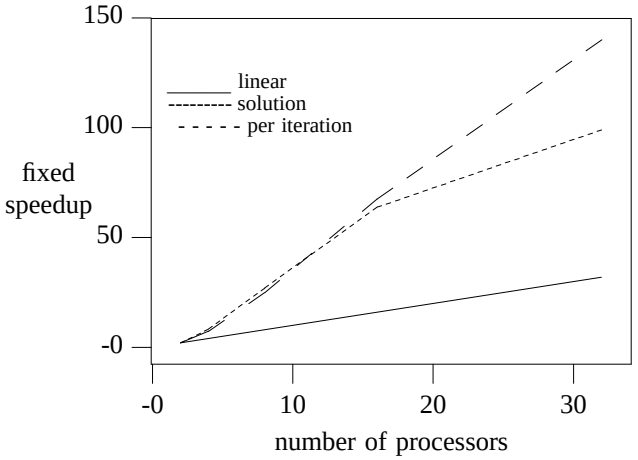

Figure 7.1. Measured speedup (relative to 2 processors) for the solution and per iteration times of the SC-PCG-QSC method on the iPSC/2 for up to 32 processors configurations and fixed 94×94 grid.

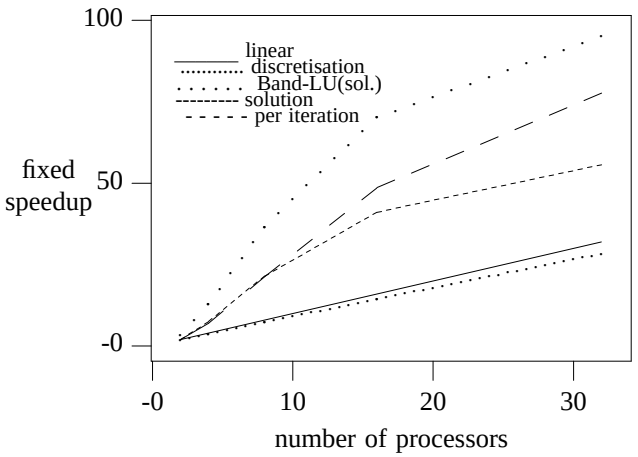

Figure 7.2. Measured speedup (relative to 2 processors) for the discretisation, solution and per iteration times of the SC-PCG-QSC method on the iPSC/2 for up to 32 processors configurations and fixed 62×62 grid.

The efficiencies corresponding to these graphs vary from 88% to 93% for the discretisation process, from 100% to 398% for the solution process and from 100% to 438% for the iteration process (!). This superlinear speedup is due to both the algorithm "speedup" (as was shown in Section 5.3.4 for the serial execution of the algorithm), as well as to the speedup from increasing the number of processors. The reason why the per iteration efficiency can become so large is that the per iteration process does not suffer from the increase of the number of iterations with the number of subdomains/processors, thus taking full advantage of the merits of the domain decomposition. It is our understanding that this "speedup" is not the standard way to measure the efficiency of our algorithm and does not reflect the parallelisation of the computation only, but the merits of the domain decomposition ordering too (mainly the decrease of the bandwidth of the blocks with the number of processors in one direction).

Figure 7.2 also graphs the "speedup" of SC-PCG-QSC with respect to Band-LU which we expect to be superlinear as well. The discretisation process time complexity is not affected by the domain decomposition, so its efficiency is in the "acceptable" ranges. Its degradation from the linear is mainly due to the more complicated code for the subdomain collocation than for the natural ordering collocation, and due to the non-scaling of the size of the problem with the number of processors, and the imperfect load balance.

Finally, in Table 7.1 we include some of the numerical data that was used to draw Figures 7.1 and 7.2. This table has also a column for the memory requirements of the two methods/solvers. The memory requirements of the SC-PCG-QSC method are far less than those for Band-LU. The memory requirements of SC-PCG-QSC decrease with the number of processors. More specifically they decrease with the number of processors in one direction (p_n) which affects the bandwidth of the blocks D_i and with the total number of processors, since the data is split among processors. We were unable to run Band-LU on the system of the 94×94 grid due to the memory limitations of the iPSC/2 node.

Table 7.1. Time in milliseconds on the iPSC/2 for the discretisation, solution and iteration processes of the Band LU factorisation and SC-PCG-QSC algorithms for various grid sizes and processor configurations. A ‘‘-’’ means not applicable. The memory requirements are in floating-point numbers.

method	$n+1$	P	p_n	p_m	discret.	solution	per iter.	iteration	memory
Band-LU	46	1	1	1	29618	58534	-	-	238572
SC-PCG-		2	2	1	16031	19370	655.07	10481	68349
-QSC		4	4	1	8233	5377	195.55	3911	21225
		8	8	1	4148	2257	74.65	1941	7815
		16	8	2	2122	1401	42.85	1243	3971
		32	8	4	1090	1018	27.60	938	2015
Band-LU	62	1	1	1	54076	182604	-	-	555660
SC-PCG-		2	2	1	29104	53508	1503.70	27067	153605
-QSC		4	4	1	14849	14333	427.00	10248	45333
		8	8	1	7458	4988	140.90	4227	15389
		16	16	1	3738	2601	61.85	2412	6465
		32	16	2	1912	1921	38.75	1821	3277
	94	2	2	1	66899	239333	4935.95	113527	491541
		4	4	1	33915	56754	1331.00	38599	137517
		8	8	1	17030	17672	396.65	14676	42585
		16	16	1	8536	7508	146.20	6871	15639
		32	32	1	4280	4830	70.50	4653	7296
			16	2	4328	5019	81.00	4698	7883

7.2. A non one-to-one mapping of subdomains to processors.

So far we have assumed that there is an one-to-one mapping of subdomains to processors. In this section we present performance evaluation results for other mappings of subdomains and of the corresponding computation to processors. In this way we are able to measure the speedup and efficiency of our implementation in the standard way. In Figure 7.3 we fix the grid to 94×94 and the domain decomposition to 32×1 subdomains and increase the number of processors from 1 to 16 so that the number of subdomains assigned to a processor ranges from 32 to 2 respectively. We then measure the speedup of SC-PCG-QSC algorithm as the ratio of the time for the algorithm to run on one processor over the time to run on P processors, for the same grid size and domain decomposition.

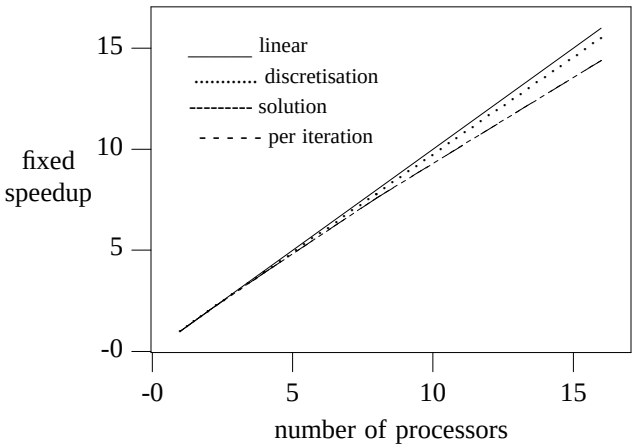

Figure 7.3. Measured speedup for the discretisation, solution and per iteration times of the SC-PCG-QSC method on the iPSC/2 for up to 16 processors configurations and fixed 94×94 grid and 32×1 domain decomposition.

Table 7.2. Time in milliseconds on the iPSC/2 for the discretisation, solution and iteration processes of the SC-PCG-QSC algorithm for various grid sizes, domain decompositions and processor configurations. The memory requirements are in floating-point numbers.

$n+1$	$P=p_n$	$D=d_n$	discret.	solution	per iter.	iteration	memory
62	1	8	57184	37391	1046.30	31389	120943
	2		29056	19021	533.70	16011	60947
	4		14848	9643	271.10	8133	30575
	8		7446	4986	140.90	4227	15389
	1	16	57188	34325	807.40	31489	99647
	2		29056	17545	413.30	16119	50299
	4		14848	8981	211.80	8260	25251
	8		7448	4716	111.50	4348	12727
	16		3738	2604	61.90	2414	6465
94	1	8	132136	137448	3071.75	113655	337455
	2		66786	69227	1549.05	57315	169443
	4		33912	34899	781.40	28912	84871
	8		16998	17633	395.65	14639	42585
	1	16	132138	110659	2141.95	100672	244607
	2		66788	55949	1083.85	50941	123019
	4		33912	28283	548.25	25768	61659
	8		16998	14383	279.15	13120	30979
	16		8536	7505	146.10	6867	15639
	1	32	132140	125880	1827.65	120625	223071
	2		66788	63880	927.60	61222	112251
	4		33912	32361	470.20	31033	56275
	8		16996	16562	240.75	15890	28287
	16		8536	8735	127.15	8392	14293
	32		4280	4830	70.50	4653	7296

The efficiencies corresponding to the speedups of Figure 7.3 range from 97% to 99% for the discretisation process, and from 90% to 98% for the solution and per iteration processes. The discretisation process speedup is almost linear, since the same (complicated) code is used on one processor and on P processors. It slightly degrades from the linear due to the overhead of non-scaling the problem size with the number of processors and the imperfect load balancing. The per iteration process speedup is also very close to linear. It degrades from the linear for the same reasons as the discretisation process and in addition it suffers from synchronisation overhead. The same holds for the solution process speedup, since the number of iterations is not affected by the number of processors, as in the case of the one-to-one mapping of subdomains to processors in Section 7.1.

Table 7.2 lists some timing results from non one-to-one mapping of D subdomains to P processors. We assume a $d_n \times 1$ domain decomposition and a $p_n \times 1$ processor layout.

7. References.

[Ahlb75] **Ahlberg, J. H.** and **T. Ito,** *A collocation method for two-point boundary value problems,* Math. Comp., **29** (1975), pp. 129–131, 761–776.

[Arch73] **Archer, D. A.,** *Some collocation methods for differential equations,* Rice University, Houston, TX, Ph.D. thesis, 1973.

[Axel84] **Axelsson, O.** and **V. A. Barker,** *Finite element solution of boundary value problems,* Academic Press, 1984.

[Bjor86] **Bjorstand, P. E.** and **O. B. Widlund,** *Iterative methods for the solution of elliptic problems on regions partitioned into substructures,* SIAM J. Numer. Anal., **23** (1986), pp. 1097–1120.

[Bram86] **Bramble, J. H., J. E. Pasciak** and **A. H. Schatz,** *The construction of preconditioners for elliptic problems by substructuring I,* Math. Comp., **47** (1986), pp. 103–134.

[Bram88] **Bramley, R.** and **A. Sameh,** *A robust parallel solver for block tridiagonal systems,* Proceedings of the 1988 International Conference on Supercomputing (ICS88), July 1988, St. Malo, France.

[Cave72] **Cavendish, J. C.,** *A collocation method for elliptic and parabolic boundary value problems, using cubic splines,* Univ. of Pittsburgh, PA, Ph.D. thesis, 1972.

[Chan87] **Chan, T. F.,** *Analysis of preconditioners for domain decomposition,* SIAM J. Numer. Anal., (1987), pp. 382–390.

[Chri88a] **Christara, C. C., E. N. Houstis** and **J. R. Rice,** *A Parallel Spline Collocation − Capacitance Method for Elliptic PDEs,* Proceedings of the 1988 International Conference on Supercomputing (ICS88), July 1988, St. Malo, France.

[Chri88b] **Christara, C. C.,** *Spline collocation methods, software and architectures for linear elliptic boundary value problems,* Ph.D. thesis, Purdue University, IN, U.S.A., 1988.

[Chri89] **Christara, C. C.** and **E. N. Houstis,** *A domain decomposition spline collocation method for elliptic partial differential equations,* Proceedings of the fourth Conference on Hypercubes, Concurrent Computers and Applications (HCCA4), March 1989, Monterey, CA, U.S.A.

[Chri90a] **Christara, C. C.,** *Conjugate Gradient Methods for Spline Collocation Equations,* to appear in Proceedings of the fifth Distributed Memory Computing Conference (DMCC5), April 1990, Charleston, SC, U.S.A.

[Chri90b] **Christara, C. C.,** *Quadratic Spline Collocation Methods for Elliptic Partial Differential Equations,* Univ. of Toronto, DCS Tech. Rep. (1990), submitted for publication.

[Chri90c] **Christara, C. C.,** *Iterative Solution of Spline Collocation Equations,* Univ. of Toronto, DCS Tech. Rep., in preparation.

[Conc76] **Concus, P., G. H. Golub** and **D. O'Leary,** *A generalised Conjugate Gradient method for the numerical solution of elliptic Partial Differential Equations,* Sparse Matrix Computations, Bunch J. and D. Rose (eds.), Academic Press, New York 1976, pp. 309–322.

[Conc85] **Concus, P., G. H. Golub** and **G. Meurant,** *Block Preconditioning for the Conjugate Gradient method,* SIAM J. Sci. Statist. Comput., **6** (1985), pp. 220–252.

[Dani75] **Daniel, J. W.** and **B. K. Swartz,** *Extrapolated collocation for two point boundary value problems using cubic splines,* J. Inst. Maths Applics, **16** (1975), pp. 161–174.

[Dryj84] **Dryja, M.,** *A finite element − capacitance method for elliptic problems on regions partitioned into subregions,* Numer. Math., **44,** (1984), pp. 153–168.

[Dryj86] **Dryja, M.** and **W. Proskurowski,** *Iterative methods in subspaces for solving elliptic problems using domain decomposition,* University of Southern California, Tech. Rep. CRI-86-10 (1986).

[Fyfe68] **Fyfe, D. J.,** *The use of cubic splines in the solution of two-point boundary value problems,* Comput. J., **17** (1968), pp. 188–192.

[Golu85] **Golub, G. H.** and **C. F. van Loan,** *Matrix computations,* John Hopkins University Press, 1985 (476 pgs).

[Gust88] **Gustafson, J. L., G. R. Montry** and **R. E. Benner,** *Development of Parallel Methods for a 1024-Processor Hypercube,* SIAM J. Sci. Statist. Comput., **9,** 4 (1988), pp. 609–638.

[Hage81] **Hageman, L. A.** and **D. M. Young,** *Applied Iterative Methods,* Academic Press 1981.

[Hest52] **Hestenes, M.** and **E. Stiefel,** *Methods of Conjugate Gradients for solving linear systems,* J. Res. Natl. Bur. Stand. Sect B, **49** (1952), pp. 409–436.

[Hous87] **Houstis, E. N., E. A. Vavalis** and **J. R. Rice,** *Convergence of an $O(h^4)$ cubic spline collocation*

method for elliptic partial differential equations, SIAM J. Numer. Anal., **25** (1988), pp. 54–74.

[Hous88a] **Houstis, E. N., J. R. Rice, C. C. Christara** and **E. A. Vavalis,** *Performance of scientific software*, Mathematical Aspects of Scientific Software, (J. R. Rice, ed.), Springer Verlag, 1988, pp. 123–156.

[Hous88b] **Houstis, E. N., J. R. Rice** and **E. A. Vavalis,** *A Schwartz Splitting variant of Cubic Spline Collocation Methods for Elliptic PDEs*, Proceedings of the third Conference on Hypercubes, Concurrent Computers and Applications, January 1988, Pasadena, CA, U.S.A.

[Inte88] **Intel Scientific Computers,** *iPSC/2 Performance report*, Intel Scientific Computers, January 1988, Beaverton, OR, U.S.A.

[Irod87] **Irodotou-Ellina, M.,** *Spline collocation methods for high order elliptic boundary value problems*, Aristotle University of Thessaloniki, Greece, Ph.D. thesis, 1987.

[Irod88] **Irodotou-Ellina, M.** and **E. N. Houstis,** *An $O(h^6)$ quintic spline collocation method for fourth-order two-point boundary value problems* BIT, **28** (1988), pp. 288–301.

[Kamm74] **Kammener, W. J., G. W. Reddien** and **R. S. Varga,** *Quadratic interpolatory splines*, Numer. Math., **22** (1974), pp. 241–259.

[Keye87] **Keyes, D. E.** and **W. D. Gropp,** *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., **8** (1987), pp. s166–s202.

[Khal82] **Khalifa, A. K.** and **J. C. Eilbeck,** *Collocation with quadratic and cubic splines*, IMA J. Numer. Anal., **2** (1982), pp. 111–121.

[OLea87] **O'Leary, D.,** *Parallel Implementation of Block Conjugate Gradient Algorithm.* Parallel Comput., **5** (1987), pp. 127–140.

[Orte88] **Ortega, J. M.,** *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, 1988 (305 pgs).

[Rodr86] **Rodrigue, G.,** *Some ideas for decomposing the domain of elliptic Partial Differential Equations in the Schwarz process*, Commun. Appl. Numer. Method, **2** (1986), pp. 245–249.

[Russ72] **Russell, R. D.** and **L. F. Shampine,** *A collocation method for boundary value problems*, Numer. Math., **19** (1972), pp. 1–28.

[Saka83] **Sakai, M.** and **R. Usmani,** *Quadratic spline solutions and two-point boundary value problems*, Publ. RIMS, Kyoto University, **19** (1983), pp. 7–13.

[Tang87] **Tang, Wei-Pai,** *Schwartz Splitting and Template Operators*, Stanford University, Ph.D thesis, 1987.

[Varg62] **Varga, R. S.,** *Matrix iterative analysis*, Prentice Hall, 1962.

[Youn71] **Young, D. M.,** *Iterative Solution of Large Linear Systems*, Academic Press 1971.

Acknowledgements

I wish to thank Elias N. Houstis and the reviewers for their suggestions for improving this manuscript. I also wish to thank the Cornell Theory Center for letting me use their iPSC/2 hypercube for free!