Conjugate Gradient Methods for Spline Collocation Equations

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

Abstract

We study the parallel computation of linear second order elliptic Partial Differential Equation (PDE) problems in rectangular domains. We discuss the application of Conjugate Gradient (CG) and Preconditioned Conjugate Gradient (PCG) methods to the linear system arising from the discretisation of such problems using quadratic splines and the collocation discretisation methodology. Our experiments show that the number of iterations required for convergence of CG-QSC (Conjugate Gradient applied to Quadratic Spline Collocation equations) grows linearly with the square root of the number of equations. We implemented the CG and PCG methods for the solution of the Quadratic Spline Collocation (QSC) equations on the iPSC/2 hypercube and present performance evaluation results for up to 32 processors configurations. Our experiments show efficiencies of the order of 90%, for both the fixed and scaled speed-ups.

1. Introduction.

We study the parallel solution of the Partial Differential Equation (PDE) problem

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

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

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

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 this paper we discuss the application of Conjugate Gradient (CG) and Preconditioned Conjugate Gradient (PCG) methods to the linear system arising from the discretisation of the above problem using quadratic splines and the collocation discretisation methodology. 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 (SC) are known to give rise to similar structure linear systems.

We implemented the CG and PCG methods for the solution of the Quadratic Spline Collocation (QSC) equations on the iPSC/2 hypercube and present performance evaluation results for up to 32 processors configurations. The implementation can be straightforward extended to several other MIMD architectures, including linear array, 2-dimensional grid of processors, as well as shared memory machines.

The methods for the parallel computation of PDEs can be classified in 3 general groups: the *domain decomposition* or *substructuring* methods, in which we assume the decomposition of the domain of problem definition into non-overlapping subdomains, the *domain splitting* methods, in which the domain is decomposed into overlapping subdomains, and those methods that directly use the parallelism involved in the process of solving the linear system arising from the discretisation of the PDE problem.

In [Chri90a], [Chri89], [Chri88a] we have studied domain decomposition methods for solving the above PDE problem and presented the results from the implementation of those on the iPSC/2, NCUBE/7 and SEQUENT BALANCE 21000 parallel machines. In [Hous88b] domain splitting methods are integrated with cubic spline collocation and implemented on the NCUBE/7 hypercube. This paper falls in the third category of methods for the parallel computation of PDEs.

Many researchers have studied the convergence and/or parallel implementation of CG and PCG methods applied to systems arising from the discretisation of elliptic problems by other Finite Element Methods (FEMs), or to the Schur complement systems arising from domain decomposition methods and appropriate reordering [Keye87], [Dryj84], [Bram86], [Bjor86], [Dryj86]. Others [Rodr86], [Tang87] experiment with domain splitting methods. The study of the solution of SC equations is limited due to the fact that the development of optimal schemes for two-dimensional problems is very recent [Hous87], [Irod87], [Chri88b], 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 suc-

cessful study of the application of CG methods to SC 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] so 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) in the case that $\alpha \cdot \beta = 0$, ie. the boundary operator is either Dirichlet or Neumann. 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. The one-dimensional quadratic splines are constructed so that the boundary operator equation (1.2) is satisfied exactly at any point on $\partial\Omega$.

We define the set of collocation points T_Δ to be the set of midpoints of all subrectangles of Δ. Note that all the collocation points lie in the interior of Ω. We determine the quadratic spline approximation $v \in S_{2,\Delta}$ to u in two steps by the following equations:

Step 1: $Lv = g$ on T_Δ (2.1)

Step 2: $Lu_\Delta = g - P_L v$ on T_Δ (2.2)

where P_L is appropriate perturbation operator, defined in [Chri88b], [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 tridiagonal linear system, of $n \cdot m$ equations. If we assume that the ordering of the collocation points is bottom-up and then left-to-right every block is of order m, the upper and lower bandwidth is $m+1$ and there are n blocks on the diagonal. Figure 2.1 shows the pattern of non-zero entries in the QSC matrix.

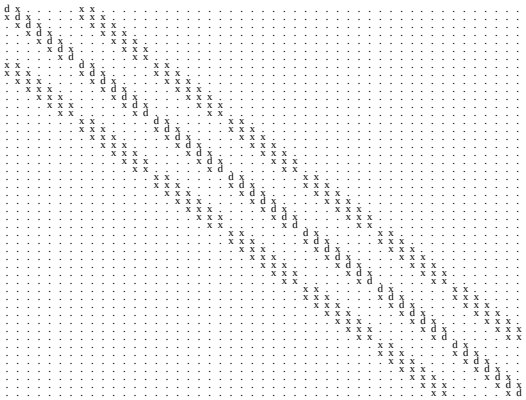

Figure 2.1. Structure of the matrix of QSC equations for $n = m = 7$. x denotes a non-zero off-diagonal element, d a non-zero diagonal one, while all zero entries are represented by character ‘.’.

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 number 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 $||r^{k-1}|| \leq \varepsilon$ (or $||r^{k-1}|| \leq ||r^0|| \varepsilon$) 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.

The CG method without preconditioner follows similar steps, with the exception that M is assumed to be the identity matrix, so the back-and-forward substitutions are avoided. Also, the computation of an inner product can be avoided, when using the Euclidean norm in line 3 of the algorithm.

4. The CG and PCG methods for the QSC equations.

We first experimented with the convergence of the CG iterations applied to the QSC equations. The results show that the number of CG iterations required to satisfy a stopping criterion as in line 3 of the PCG algorithm grows linearly with the square root of the order of the system. In the case where $n = m$ the order of the system is $O(n^2)$, so the number of iterations grows linearly with n. Table (4.1) shows the number of iterations required for the CG-QSC method (CG method applied to QSC equations) when applied to the problem

$$u_{xx} + u_{yy} = f \quad \text{in } \Omega = (0,1)\times(0,1) \qquad (4.1a)$$

$$u = g \quad \text{in } \partial\Omega. \qquad (4.1b)$$

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 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 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}. Figure 4.1 shows graphically the data of Table 4.1.

Table 4.1. Number of iterations required for the convergence of the CG method applied to the QSC equations (2.1)-(2.2) for several grid sizes.

grid size	number of	number of iterations	
$n+1$	equations	step 1	step 2
5	16	5	5
9	64	11	10
17	256	24	19
25	576	36	26
33	1024	49	34
41	1600	63	42
49	2304	77	50
57	3136	91	58
65	4096	105	66

It is interesting to note that the number of iterations required for convergence of step 2 is exactly $n+1$ (or $n+2$) while the slope of the number of iterations curve required for convergence of step 1 is about 1.6875.

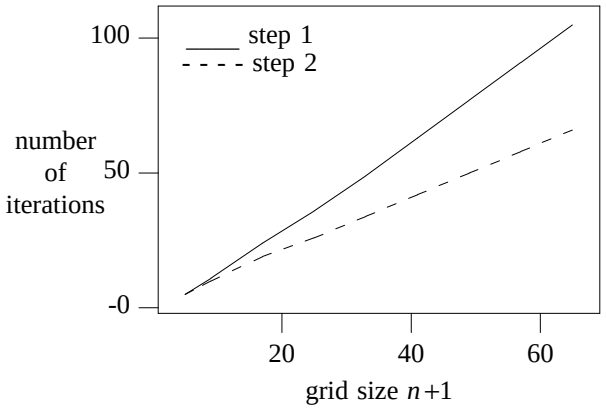

Figure 4.1. Plot of the number of iterations required for the convergence of the CG method when applied to QSC equations for Problem 4.1 versus the grid size in one dimension, for both steps of the QSC method.

In an attempt to explain why the number of iterations required for the convergence of the CG-QSC method grows proportionally with n, we consider a Helmholtz problem ($b = d = e = 0$) with constant coefficients (a, c, f constants) and Dirichlet boundary conditions ($\beta = 0$) and state a theorem which is proved in [Chri90b].

Theorem. *Under the assumptions that* $a, c > 0$ *and* $\pi^2\left(\frac{a}{(bx-ax)^2} + \frac{c}{(by-ay)^2}\right) > f$, *the spectral norm of the inverse of the matrix of QSC equations in the case of a Helmholtz problem with constant coefficients and Dirichlet boundary conditions is bounded, as* $n \to \infty$, $m \to \infty$.

A similar theorem holds in the case of Neumann boundary conditions. Taking in account that the norm of the matrix of QSC equations grows proportionally with n^2, we conclude that the condition number of the matrix of QSC equations also grows proportionally with n^2. For a symmetric positive definite system $Ax = b$ we know [Axel84] that the number of CG iterations required for convergence grows proportionally with the square root of the condition number of A. For the case of a Helmholtz problem with constant coefficients and Dirichlet boundary conditions the matrix of QSC equations is symmetric and positive definite, so the number of iterations required for the convergence of the CG-QSC method grows proportionally with n. Figure 4.2 shows the behaviour of the residual of the QSC system as the CG iterations proceed. It is interesting to note that for PDE problems other than the Helmholtz problem, for

which the QSC equations are not symmetric we have successfully applied the CG method and its asymptotic behaviour was not extremely different from the one for Problem 4.1.

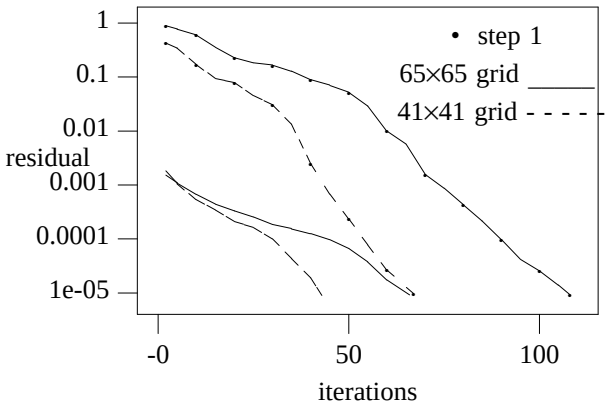

Figure 4.2. Plot of the residual of the QSC system versus the iteration number of the CG algorithm for Problem 4.1, for several grid sizes and for both steps of the QSC method. The residual is in log scale.

We also experimented with the performance of the CG-QSC method as compared with solving the QSC equations with standard banded LU factorisation (Band-LU). Figure 4.3 shows graphically the results. The slope of the curve of the solution time versus the grid size corresponding to the CG-QSC method is clearly lower than the one for Band-LU. This agrees with the theoretically expected performance of the two methods, since Band-LU is $O(n^4)$, while CG-QSC is $O(n^3)$, where we have again assumed that $n = m$. Note that this holds for step 1 of the QSC method. For step 2 both Band-LU and CG methods are $O(n^3)$, assuming the factorisation of the matrix of QSC equations is saved from step 1.

The CG method is not the only iterative solver that is faster than direct band solvers for the QSC equations. In [Hous88a] we experiment with several iterative solvers, that outperform the direct ones in both time and memory requirements.

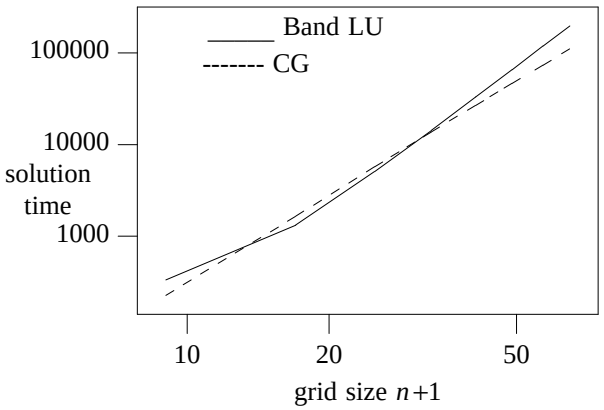

Figure 4.3. Log-log scale plot of the time in milliseconds for the solution of step 1 of the QSC equations with the CG and Band-LU methods versus the grid size in one dimension. The computation was carried out on one processor of the iPSC/2 hypercube.

For matrices of block tridiagonal structure it is quite common to choose as preconditioner the tridiagonal part of the original matrix, in order to accelerate the convergence rate of the CG method. In the case of QSC equations the tridiagonal part T of the matrix is also block-diagonal. When the CG method is applied to the QSC equations arising from the discretisation of a PDE problem with operator other than the Laplace operator, our experiments show that using T as preconditioner accelerated the convergence of the CG method. In [Chri90c] we study the construction of appropriate preconditioners for the QSC equations. In the rest of the paper any reference to the CG-QSC method will assume that an appropriate preconditioner is used whenever necessary.

5. Implementation of the CG-QSC method on hypercube architectures.

In this section we discuss in more detail how the computation involved in the CG-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.

5.1. 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 to distribute 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 CG-QSC method the distribu-

tion of data is motivated 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 by rows in a sparse matrix storage scheme, so that only the non-zero elements of every row are stored. Every processor holds the rows corresponding to one or more blocks in the block notation of the matrix. For simplicity we assume the number of processors P divides n exactly. So every processor will store $\frac{nm}{P}$ equations. Also every processor will store the respective rows of the vectors r, z and x. As far as the direction vector p is concerned a processor will update those components corresponding to the rows of A it stores, but will have storage for the "neighbouring" components, more specifically m positions on the top of the part it is going to update and m positions at the bottom.

5.2. 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 QSC with the midpoints as collocation points, and a bottom-up left-to-right numbering of them, the parallel generation of the matrix A can be viewed as a *line collocation* method. Every processor generates the rows of A it is assigned to, that is, the equations corresponding to one or more vertical grid lines, with no need to communicate with any other processor.

5.3. Computing the product of A by a vector.

According to the above assignment of the elements of A and p to the processors, a processor will compute the inner product of the rows of A it holds with the vector p. Due to the block-tridiagonal structure of the matrix A any processor needs to receive at most $2m$ components of p by other processors, the rest reside already in the local memory of the processor. This fact has two nice effects: First, that only neighbour communication is necessary, if we assume that the assignment of blocks of rows of A to the processors is done according to the standard gray code ordering of the processors. Second, that the amount of data transfer per processor does not grow with the number of processors. It only grows with m, the grid size in one direction. This helps so that the speedup does not degrade much as the number of processors increases.

5.4. Preconditioning.

The assignment of the blocks of T to the processors is similar to that of A. Since T is block diagonal, every processor can work independently for the back-and-forward substitutions of the blocks of T it is assigned to. So preconditioning with T does not increase the communication overhead.

5.5. 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.

6. Performance results.

In this section we discuss the performance of the CG-QSC method on various processor configurations of the iPSC/2 hypercube. We refer to the basic computational constructs of the CG-QSC method as *discretisation process, solution process* and *per iteration process.* The per iteration process includes the computation of one (P)CG iteration, while the solution process includes the factorisation of the preconditioner (if there is one) and the computation of all iterations.

6.1. Speedup and efficiency.

We first measure the so called *scaled speedup* [Gust88], [Orte88]. According to the definition of scaled speedup we need to choose a different size problem for each processor configuration. For the solution process of the CG-QSC method, the problem size, that is, the operation counts, is $O(n^3)$, while for each iteration it is $O(n^2)$, as it is for the discretisation process. More specifically, we scale the problem size as follows: Let n be the number of grid points in one dimension, for which we let the CG-QSC program to run on a single processor. We then choose n_P to be such that $n_P^3 = Pn^3$ and let the CG-QSC program to run on P processors for a grid size n_P. Then the scaled speedup for the (solution process of the) CG-QSC algorithm is $\frac{t_1(n)}{t_P(n_P)} \cdot P$, where $t_i(j)$ is the time elapsed for the execution of the program on i processors and grid size j in one dimension. Similarly for the discretisation process and the per iteration computation we choose n_P such that $n_P^2 = Pn^2$. Alternatively, we can compute the scaled speedup for 2 different grid sizes n and n_P as $\frac{t_1(n)}{t_P(n_P)} \cdot \frac{n_P^3}{n^3}$, for the solu-

tion process of the CG-QSC algorithm and as $\frac{t_1(n)}{t_P(n_P)} \cdot \frac{n_P^2}{n^2}$, for the discretisation and per iteration processes.

In Figure 6.1 we plot the estimated scaled speedup for the discretisation, solution and per iteration processes of the CG-QSC algorithm. The grid sizes for this plot vary from 25 for a single processor to 97 for 32 processors. The discretisation process does not suffer from any communication overhead and the slight degradation of the speedup away from the linear one is due to duplicate computations done in all processors, in order to initialise certain parameters of the problem, as well as to a few differences in the code for a single processor from that for multiple processors. The speedup curves for the solution and per iteration processes look very similar. The degradation of speedup in these cases is mainly due to communication overhead as well as synchronisation and load balancing. We would like to point out that the CG-QSC algorithm is perfectly load balanced as far as computation is concerned. Communication is also well load balanced with the exception of the nearest neighbour communication, that is required for the computation of the matrix-vector product *Ap*, in which the first and last processors remain idle, during the time the others exchange *m* components of a vector. Also, the unreliability of the hardware might cause some load imbalance.

Based on the speedups plotted in Figure 6.1 the efficiency of the discretisation process ranges from 90% to 98%, while the efficiency of the solution and per iteration processes range from 79% to 93%.

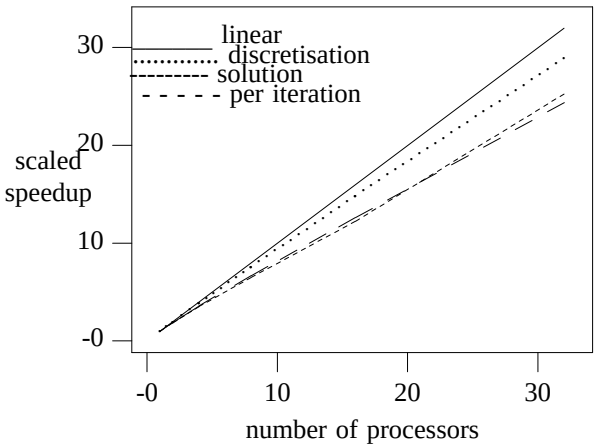

Figure 6.1. Measured speedup for the discretisation, solution and per iteration times of CG-QSC on the iPSC/2 for up to 32 processors configurations.

We next plot the *fixed speedup* for grid sizes 97 and 65 in Figures 6.2 and 6.3 respectively. This turns out to be better than the scaled one for small number of processors, but degrades faster for large number of processors. This comes from the fact that the fixed speedup suffers from the overhead of carrying out small amount of computation in each processor. It is clear that the slope of the fixed speedup curve for large number of processors is lower than that of the scaled one, and that the 65 grid size speedup is worse than the 97 grid size one.

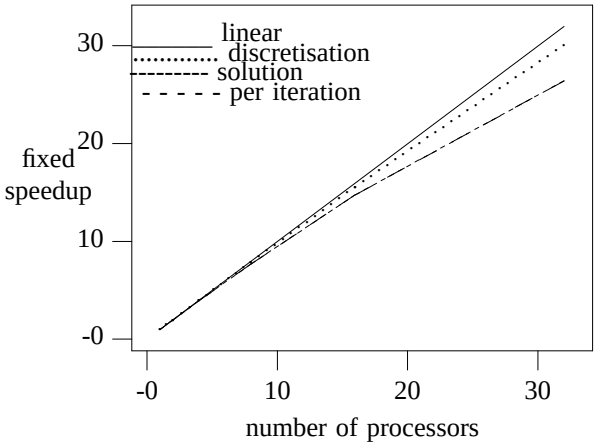

Figure 6.2. Measured speedup for the discretisation, solution and per iteration times of CG-QSC on the iPSC/2 for up to 32 processors configurations and fixed 97×97 grid.

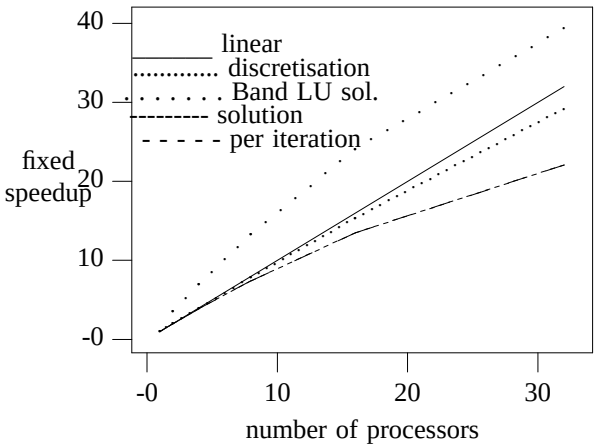

Figure 6.3. Measured speedup for the discretisation, solution and per iteration times of CG-QSC on the iPSC/2 for up to 32 processors configurations and fixed 65×65 grid.

It is interesting to note that the efficiencies of the processors based on the fixed speedups are 100% for 2 processors for both the grid sizes shown in the two figures. More specifically, the efficiency based on fixed speedup varies from 94% (91%) to 100% for the discretisation process and grid size 97 (65), and from 83%

(69%) to 100% for the solution and per iteration processes for the same grid size(s).

In Figure 6.3 we also plot the "speedup" of the CG-QSC solution process with respect to the Band-LU solution process carried out in a single processor. This is clearly superlinear, due to the merits of the CG-QSC method. We were unable to run the Band-LU algorithm for larger than 65 grid sizes, due to the limit in the local memory of a processor.

Finally, in Table 6.1 we include some of the numerical data that was used to draw Figures 6.1, 6.2 and 6.3. This table has also a column for the memory requirements of the two methods/solvers. The memory requirements of the CG-QSC method are far less than those for Band-LU. The memory requirements of CG-QSC decrease almost linearly with the number of processors.

Table 6.1. Time in milliseconds on the iPSC/2 for the discretisation, solution and iteration processes of the Band LU factorisation and CG-QSC algorithms for various grid sizes and processor configurations. A ‘‘-’’ means not applicable. The memory requirements are in floating-point numbers.

method	$n+1$	discret.	solution	per iteration	memory	P
Band-LU	25	8460	5048	-	35712	1
CG-QSC		8446	5716	141.30	10944	1
		4234	2958	73.10	5496	2
		2156	1664	41.20	2784	4
		1120	1028	25.50	1416	8
Band-LU	33	14952	14228	-	79872	1
CG-QSC		14948	13762	252.70	19456	1
		7492	7040	129.26	9760	2
		3792	3827	70.27	4930	4
		1952	2222	40.86	2498	8
		1020	1452	26.74	1282	16
		560	1130	20.78	674	32
Band-LU	41	23268	32960	-	150400	1
CG-QSC		5892	7164	106.27	7680	4
Band-LU	49	33360	65464	-	253440	1
CG-QSC		4296	6617	81.25	5568	8
Band-LU	65	59028	197712	-	581632	1
CG-QSC		59020	110876	1022.37	77824	1
		29560	55748	514.06	38976	2
		14876	28425	262.12	19584	4
		7556	14926	137.65	9856	8
		3848	8210	75.76	4992	16
		2024	5022	46.35	2560	32
	97	132504	349274	2306.25	175104	1
		66432	175340	1157.76	87648	2
		33292	88472	584.16	43968	4
		16820	45314	299.22	22080	8
		8504	23624	156.00	11136	16
		4400	13217	87.29	5664	32

6.2. Communication time.

As explained before, the communication overhead of the CG-QSC algorithm is due to the matrix-vector multiplication (neighbour communication) and the inner product and norm computation (global communication). In order to verify the theoretically obtained result of Section 5.3, that the communication overhead for computing the product of A by a vector does not increase with the number of processors, we attempt to measure the time spent in communication, during the computation of the product of A by a vector in the following way. We let our code run, skipping all the computation statements and executing only the *send/receive* operations of the iPSC/2 hypercube. For a better accuracy we

let it carry out several iterations and then take the average of the time elapsed. It is our understanding that in this way we include in our measurements the computation time required for addressing the message buffers and the overhead spent in synchronisation and load balancing. Table 6.2 lists the average communication time measured in this way, for several problem sizes and number of processors.

Table 6.2. Communication time in milliseconds on the iPSC/2 during the computation of the matrix-vector multiplication for various grid sizes and processors configurations.

P / $n+1$	2	4	8	16	32
25	0.64	1.28	1.28	1.28	1.28
49	1.28	2.56	2.88	2.88	2.88
81	1.28	2.88	2.88	2.88	2.88

From the results of Table 6.2 it is clear that the communication overhead of our implementation is not affected by the number of processors, except in the case of 2 processors, in which there is only one-way communication. We find these timings quite consistent with our theoretical statements taking in account the factors of clock accuracy and unreliability of the hardware. We also note that the communication overhead is affected (not necessarily linearly) by the size of the problem. This agrees quite well with the communication performance report for the iPSC/2 hypercube [Inte88], where it is stated that the time for node-to-node communication is about the same for 0-100 bytes messages (up to 25×25 grid), it is about double for a message of 104 bytes length, than for one of 100 bytes length, and varies slightly for messages of 104-1024 bytes length (this includes the largest grid size, for which the CG-QSC method was tested).

We have carried out similar experiments in order to measure the time spent in communication during the computation of the inner products and norms in every iteration of the PCG algorithm. Our experiments show that this time increases linearly with the dimension of the hypercube ($\log(P)$), as expected. Based on our experiments the global fan-in summation of the partial inner products takes 2-4 milliseconds for 2-32 processors configurations. Taking in account that every PCG iteration requires 3 times this type of global communication and once the neighbour communication for the matrix-vector multiplication we conclude that the time spent in communication is less than 1% of the total time for the case of 97×97 grid and 2 processors, while it is about 16% of the total time for the same grid size and 32 processors. This means that almost all what is lost in efficiency is due to communication overhead, and it leads us to suggest (once again!) that in order to benefit from the use of a lot of processors, we have to solve problems of appropriately large size.

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.,** *Schur complement preconditioned conjugate gradient methods for spline collocation equations,* to appear in Proceedings of the 1990 International Conference on Supercomputing (ICS90), June 1990, Amsterdam, the Netherlands.

[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 for his suggestions for improving this manuscript. I also wish to thank the Cornell Theory Center for letting me use their iPSC/2 hypercube for free!