Domain Decomposition and Incomplete Factorisation Methods for Partial Differential Equations

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

Abstract

In this paper we develop and study a method which tries to combine the merits of Domain Decomposition (DD) and Incomplete Cholesky preconditioned Conjugate Gradient method (ICCG) for the parallel solution of linear elliptic Partial Differential Equations (PDEs) on rectangular domains. We first discretise the PDE problem, using Spline Collocation, a method of Finite Element type based on smooth splines. This gives rise to a sparse linear system of equations. The ICCG method provides us with a very efficient, but not straightforward parallelisable linear solver for such systems. On the other hand, DD methods are very effective for elliptic PDEs. A combination of DD and ICCG methods, in which the subdomain solves are carried out with ICCG, leads to efficient and highly parallelisable solvers. We implement this hybrid DD−ICCG method on a hypercube, discuss its parallel efficiency, and show results from experiments on configurations with up to 32 processors. We apply a totally local communication scheme and discuss its performance on the iPSC/2 hypercube. A similar approach can be used with other PDE discretisation methods.

1. Introduction.

We consider the PDE problem

$$\mathbb{L}u \equiv a\mathbb{D}_x^2 u + b\mathbb{D}_x\mathbb{D}_y u + c\mathbb{D}_y^2 u + d\mathbb{D}_x u + e\mathbb{D}_y u + fu = g \quad \text{in } \Omega \equiv (ax,bx)\times(ay,by) \tag{1.1}$$

$$\mathbb{B}u \equiv \alpha u + \beta\mathbb{D}_n u = \gamma \quad \text{on } \partial\Omega \equiv \text{boundary of } \Omega \tag{1.2}$$

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

An *optimal order of convergence* discretisation method (referenced thereafter as Quadratic Spline Collocation or QSC) based on smooth quadratic splines and the collocation methodology was proposed in [Chri90a] and its convergence analysis was presented. In [Chri90c] a DD method based on the capacitance (or Schur Complement) Method (SCM) was formulated, and implemented on a hypercube architecture machine. Its computational complexity was discussed and numerical experiments were shown, which verify the theoretical behaviour of the method. In Section 2 we summarise the above results, which will be used as a background for subsequent sections.

In Section 3 we study the application of ICCG method to the system of QSC equations with natural ordering. Our experiments show that the number of iterations required for convergence of ICCG on QSC equations grows linearly with the square of the logarithm of the number of equations, thus providing us with a very efficient linear solver of computational complexity $O(n^2\log^2(n))$, where n is the number of discretisation points in one direction.

The ICCG method is unfortunately not fully parallelisable. It has a rather limited degree of parallelism, while DD methods are inherently parallel. In Section 4, we propose an integration of DD and ICCG, in which the subdomain solves required repeatedly for the solution of the capacitance system are carried out by ICCG. The proposed method has higher computational requirements than the ICCG applied to the QSC equations, but it is highly parallelisable since it is based on the DD technique and the subdomain solves are carried out on individual processors. Numerical results show the serial efficiency of the method.

In Section 5, we discuss the parallel implementation of the proposed method on a distributed memory machine and its parallel computational complexity, as well as its memory requirements. We study in more detail the communication complexity of the proposed method, when implemented on a hypercube architecture machine. Results from experiments on a 32-processor iPSC/2 hypercube show that the communication overhead is minimal, thus leading to a parallel efficiency of the order of 90%. Finally in Section 6, we briefly mention the results of a comparison between DD methods and Schwartz Splitting (SS) methods.

Many researchers have considered DD methods for the (parallel) solution of elliptic PDEs, [Keye87], [Dryj84], [Bram86], [Bjor86], [Dryj86]. Others [Rodr86], [Tang89] experiment with domain splitting methods, which assume partial overlapping of the subdomains. In [Keye87] it is stated that the subdomain solves need not always be carried out exactly. [Tang90] has proposed iterative solution of the subdomains with limited accuracy. This paper is a successful attempt to employ fast iterative solvers on QSC equations. A similar approach can be used for linear systems arising from the discretisation of other Finite Element or Finite Difference Methods.

2. Background.

2.1. The quadratic spline collocation method.

The formulation of the QSC method and its convergence analysis are extensively studied in [Chri90a]. Its experimental performance evaluation is discussed in [Hous88a]. Here we briefly recall that the *two-step* QSC equations in the case that $\alpha\cdot\beta = 0$, (i.e. the boundary operator is either Dirichlet or Neumann) form a linear system, of $n\cdot m$ equations, where n and m are the number of subintervals in the x and y direction respectively, of the grid partition of Ω. If we assume that the ordering of the collocation points is bottom-up and then left-to-right the system is block tridiagonal, every block is of order m, the upper and lower bandwidth is $m+1$ and there are n blocks on the diagonal. The QSC matrix is shown to be diagonally dominant under certain conditions [Chri90a]. In addition, when the coefficients a, c, f of (1.1) are constant, $a\cdot c > 0$, $f \le 0$ ($f < 0$ if $\beta\neq 0$) and $b = d = e = 0$ the QSC matrix is symmetric positive definite. In both steps of the QSC method the QSC matrix is the same, while the right side vector of the second step is a perturbation of the right side vector of the first step. The first step solution is a second order approximation to u and the second step solution is a fourth order one.

2.2. Schur complement methods for QSC equations.

In this section we summarise the results of [Chri90c], in which a DD ordering of the QSC points/equations/unknowns was proposed, an SCM for the QSC equations was developed, and its serial and parallel performance was discussed.

We first partition the domain Ω into $d_n\times d_m$ subdomains (boxes). With this partition we obtain $(d_n-1)\cdot d_m$ and $(d_m-1)\cdot d_n$ vertical and horizontal interfaces respectively including the $(d_n-1)\cdot(d_m-1)$ cross-points.

We then number the collocation points as follows: We number the interface points first, according to the *Second Nearest Neighbour* (SNN) ordering [Chri90c] 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. The result of the domain decomposition ordering of the equations/unknowns is the so called *arrowhead matrix* [Orte88], which can be written as a 2×2 block system as in (2.1).

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \qquad (2.1)$$

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. A_{11} is a block diagonal matrix of $D \equiv d_n\cdot d_m$ blocks, each of them being an $(\frac{n+1}{d_n}-1)\cdot(\frac{m+1}{d_m}-1)$ order banded matrix with bandwidth $\frac{n+1}{d_n}$, while A_{00}, A_{01} and A_{10} are sparse matrices.

We apply the *capacitance* or *Schur Complement* method in order to solve the partitioned system (2.1). 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 $Cx_0 = w$, $C \equiv A_{00} - A_{01}A_{11}^{-1}A_{10}$,
4. Solve $A_{11}x_1 = b_1 - A_{10}x_0$.

For the solution of the *capacitance* system $Cx_0 = w$ in step 3 above we apply Preconditioned Conjugate Gradient (PCG) iterations with preconditioner the tridiagonal matrix T holding the respective entries of A_{00}. Note that steps 1 and 4 above require the solution of a block diagonal system, i.e. the solution of the blocks of A_{11}, each one of them corresponding to a subdomain solve. If we assume that the subdomain solves are carried out directly then step 1 above involves the factorisation and forward-and-back substitutions for the banded blocks. Step 4 involves only forward-and-back substitutions for the already factored blocks. In step 3 , 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. Thus, the explicit computation of the entries of C is avoided by carrying out forward-and-back substitutions for the already factored blocks in each iteration of the PCG algorithm.

Our experiments show that the number of PCG iterations applied to the capacitance system C with preconditioner T, that are required for convergence

grows linearly with the square root of the order of C, i.e. the number of interface points/equations/unknowns. The number n_0 of interface points is given by

$$n_0 = n\cdot(\mathrm{d}_m-1) + m\cdot(\mathrm{d}_n-1) - (\mathrm{d}_n-1)(\mathrm{d}_m-1) \qquad (2.2)$$

Thus the computational complexity of the method is $O(\frac{n^4}{\mathrm{d}_n^2} + nit\cdot\frac{n^3}{\mathrm{d}_n})$, where *nit* is the number of PCG iterations applied to the capacitance system $Cx_0 = w$, and where we have assumed that $n = m$ and recalled that the bandwidth of the blocks is $O(\frac{n}{\mathrm{d}_n})$. Letting $nit = O(\sqrt{n_0})$ we obtain the computational complexity of the method with respect to the 3 free parameters n, D and d_n.

$$T_{dir}(n,\mathrm{D},\mathrm{d}_n) = O(\frac{n^4}{\mathrm{d}_n^2} + [n(\mathrm{d}_n + \frac{\mathrm{D}}{\mathrm{d}_n})]^{1/2}\cdot\frac{n^3}{\mathrm{d}_n}) \qquad (2.3)$$

It is important to note that *nit* is minimised for boxwise decompositions with about equal number of subdomains in the x and y directions, i.e. $\sqrt{\mathrm{D}}\times\sqrt{\mathrm{D}}$ domain decompositions, but the computational complexity $T_{dir}(n,\mathrm{D},\mathrm{d}_n)$ of the SCM for QSC equations is minimised for stripwise decompositions, i.e. D×1 domain decompositions, due to the fact that the bandwidth of the blocks of A_{11} is minimised for stripwise decompositions.

3. The incomplete factorisation preconditioning for QSC equations.

In this section, we study the performance of the ICCG method applied to QSC equations, referenced thereafter as ICCG-QSC. Our experiments show that the number of iterations required for convergence grows linearly with the square of the logarithm of the number of equations. If for simplicity we assume that $n = m$, then the number of iterations is $O(\log^2 n)$. In Table 3.1 we show the number of iterations required for convergence of ICCG on QSC equations for the test problem

$$u_{xx} + u_{yy} = f \quad \text{in } \Omega = (0,1)\times(0,1) \qquad (3.1)$$

$$u = g \quad \text{in } \partial\Omega, \qquad (3.2)$$

where f and g are chosen so that the solution to the problem is $u(x,y) = x^{13/2}(x-1)y^{13/2}(y-1)$. For the implementation of the ICCG method we have used the University of Waterloo package [Kigh89] 'MAT1', known also as 'WATSIT'. There were two stopping criteria used any of which causes the iterative method to terminate: i) $||r^{(k)}||_2 \le \varepsilon_1 ||r^{(0)}||_2$, where $r^{(k)}$ is the residual at the k-th iteration and $||.||_2$ the Euclidean norm of a vector, and ii) $||dx^{(k)}||_\infty \le \varepsilon_2$, where $dx^{(k)}$ is the update of approximate solution at the k-th iteration. The desired precisions were chosen to be $\varepsilon_1 = 10^{-5}$ and $\varepsilon_2 = 10^{-9}$. 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 second order solution vector from step 1. The *level* [Oppe88] of incomplete factorisation was 1. A modification for row-sum equality was used. All computation was carried out in double precision on a SUN 4/60 Sparcstation. Table 3.1 also shows the number of iterations required for convergence of the Conjugate Gradient (CG) method without preconditioner applied to the QSC equations (CG-QSC).

Table 3.1. Number of iterations required for the convergence of the CG method with incomplete factorisation preconditioning and without preconditioning applied to the two-step QSC equations corresponding to Problem (3.1)-(3.2) for several grid sizes.

grid size	number of equations	number of iterations CG-QSC		ICCG-QSC	
$n+1$		step 1	step 2	step 1	step 2
5	16	5	5	2	2
9	64	11	10	4	4
17	256	24	19	7	6
25	576	36	26	9	6
33	1024	49	34	11	7
41	1600	63	42	12	8
49	2304	77	50	14	9
57	3136	91	58	15	9
65	4096	105	66	16	9
97	9216	168	100	20	10
129	16384	197	135	23	10

We note that the number of ICCG iterations required for convergence of the second step QSC equations grows linearly with $\log(n)$. Figure 3.1 shows graphically some of the data in Table 3.1.

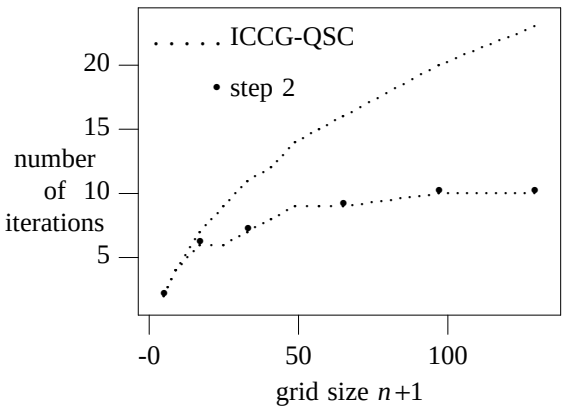

Figure 3.1. Plot of the number of iterations required for the convergence of the CG and ICCG method when applied to the two-step QSC equations for Problem (3.1)-(3.2) versus the grid size in one dimension, for both steps of the QSC method.

In Figure 3.2 we plot the time for solving the QSC equations with three different linear solvers, namely Banded Gauss Elimination (Band-LU), CG (CG-QSC) and ICCG (ICCG-QSC), versus the grid size n. The computational complexity of Band-LU is $O(n^4)$, the one of CG-QSC is $O(n^3)$, while the one of ICCG-QSC is $O(n^2\log^2(n))$. All computation was carried out in double precision on a SUN 4/60 Sparcstation.

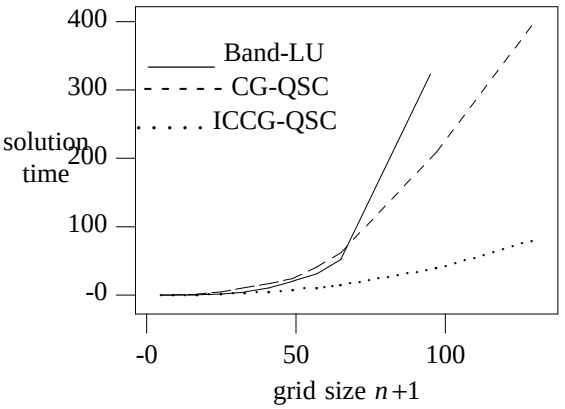

Figure 3.2. Plot of the solution time for three linear solvers when applied to the first step QSC equations for Problem (3.1)-(3.2) versus the grid size in one dimension.

Finally, in Figure 3.3, we plot the memory requirements for Band-LU, CG-QSC and ICCG-QSC in numbers (including integer and floating-point) for various grid sizes. The memory requirements for Band-LU are $O(n^3)$, while for both CG-QSC and ICCG-QSC are $O(n^2)$ as the incomplete factorisation algorithm fills a limited number of diagonals in the upper and lower part of the matrix.

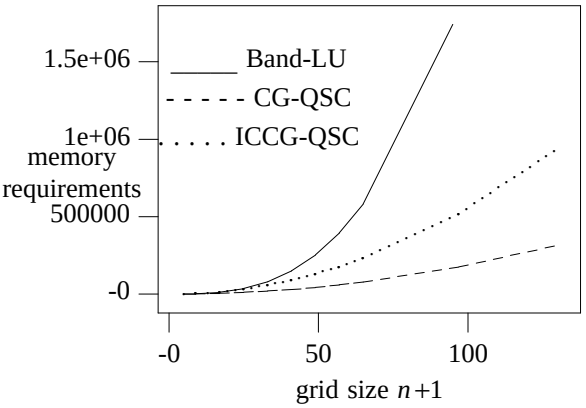

Figure 3.3. The growth of memory requirements in log. scale for three linear solvers when applied to QSC equations versus the grid size in one dimension.

4. The incomplete factorisation DD method.

From the previous section it has become clear that the ICCG method is a very fast solver for QSC equations and has moderate memory requirements. The parallelisation of this method has been studied with partial success only, as the data dependency during the process of solving for the preconditioner at every ICCG iteration allows a limited scaling of the number of processors. In [Chri91] we study the serial and parallel performance of ICCG applied to QSC equations ordered with a multicolour ordering. In this paper we do not intend to study the parallelisation of ICCG or experiment with the parallel implementation of the method on some distributed memory machine. In an attempt to take advantage of the merits of ICCG we propose a *hybrid Incomplete Factorisation Domain Decomposition* method, which will have some computational merits of both ICCG and DD and will be equally parallelisable as DD.

The method consists of four steps, similarly as the conventional SCMs.

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 $A_{11}x_1 = b_1 - A_{10}x_0$.

Here, we no longer assume that the subdomain solves are carried out by a direct solver. Rather, in step 1 above we carry out the incomplete factorisation of the blocks of A_{11}, as well as the solution by incomplete factorisation PCG. In step 3 when the PCG method with preconditioner T is applied on the capacitance system $Cx_0 = w$ and requires the computation of the product of C with a given vector, we apply inner PCG iterations for the blocks, with preconditioner the already computed incomplete factorisations of the blocks. Similarly, in step 4, the solution of the blocks is carried out by incomplete factorisation PCG. This hybrid DD-ICCG method will be referred to as *Schur Complement Incomplete Factorisation Preconditioned Conjugate Gradient* method for QSC equations (SC-IF-PCG-QSC).

4.1. Convergence of the inner iterations.

The convergence of PCG with preconditioner T applied to the capacitance system was studied in [Chri90c]. Here, we study the behaviour of the inner iterations, that is the convergence of ICCG applied to the subdomain blocks. The subdomain blocks can be viewed as banded matrices arising from the discretisation of PDE problems similar to (1.1)-(1.2) defined in the subdomains with the exclusion of the boundary. This means that the blocks possess similar properties as the original QSC equations. Our experiments show that the convergence rate of the ICCG iterations applied to

the subdomains is roughly similar to that of ICCG applied to the QSC equations. Table 4.1 shows the number of inner iterations required for convergence of ICCG on the subdomains for various grid sizes and domain decompositions. The desired precisions for the inner iterations were chosen to be $\varepsilon_1 = 10^{-4}$ and $\varepsilon_2 = 10^{-9}$, while the relative residual precision for the PCG with preconditioner T applied to the capacitance system was $\varepsilon = 10^{-5}$. There is no need for higher precision for the inner iterations than that for the PCG iteration applied to the capacitance system. It is interesting to note that the convergence is in some cases affected by the choice of domain decomposition. When the number of subdomain equations is large the choice of domain decomposition affects little the convergence of the inner iterations. When the number of subdomain equations is small, we may have the specific case where the decomposition of the domain and the numbering of the subdomain points are chosen so that the banded blocks allow very little fill-in; this is the case where the half-bandwidth of the blocks drops down to about 6. In this case the ICCG method converges in 1 iteration, i.e. it reduces to a direct method. The last 2 columns of Table 4.1 show the memory requirements of ICCG and of Band-LU applied to one subdomain block. In the case of ICCG a sparse matrix storage scheme is used and the memory requirements include floating-point numbers and integer numbers required for indexing in the sparse storage scheme. In the case of Band-LU a banded matrix storage scheme is used and the memory requirements are floating-point numbers. Figure 4.1 shows graphically the growth of the number of inner iterations required for convergence of ICCG on the subdomains versus the square of the logarithm of the number of equations of the subdomains.

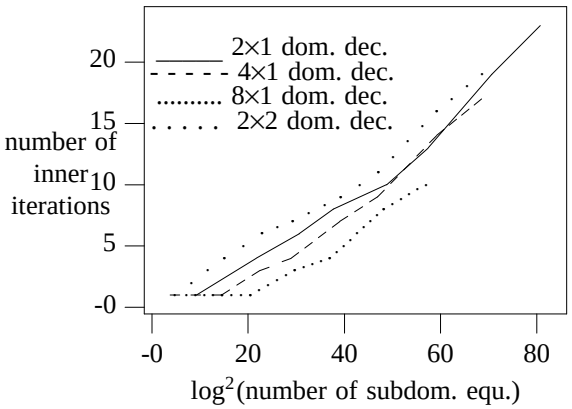

Figure 4.1. Plot of the number of inner iterations required for convergence of the CG method with incomplete factorisation preconditioning applied to QSC subdomain equations for Problem (3.1)-(3.2) versus the number of subdomain equations. The grid size $n+1$ varies from 8 to 128.

Table 4.1. Number of inner iterations required for convergence of the CG method with incomplete factorisation preconditioning applied to the QSC subdomain equations for Problem (3.1)-(3.2) for several grid sizes and domain decompositions.

domain decomp. d_n	domain decomp. d_m	grid size $n+1$	number of subdomain equations	number of inner iterations	memory requirements ICCG	memory requirements Band-LU
1	2	64	1953	12	70160	124992
1	2	96	4465	18	162016	428640
1	2	128	8001	21	291760	1024128
2	1	64	1953	13	69904	62496
2	1	96	4465	19	161632	214320
2	1	128	8001	23	291248	512064
2	2	64	961	11	34096	30752
2	2	96	2209	16	79504	106032
2	2	128	3969	19	143856	254016
1	4	64	945	8	33312	60480
1	4	96	2185	12	78328	209760
1	4	128	3937	14	142288	503936
4	1	64	945	9	32928	15120
4	1	96	2185	14	77752	52440
4	1	128	3937	17	141520	125984
2	4	64	465	7	16192	14880
2	4	96	1081	11	38440	51888
2	4	128	1953	13	70160	124992
4	2	64	465	8	16064	7440
4	2	96	1081	12	38248	25944
4	2	128	1953	15	69904	62496
1	8	64	441	5	14888	28224
1	8	96	1045	7	36484	100320
1	8	128	1905	9	67552	243840
8	1	64	441	4	14440	3528
8	1	96	1045	8	35812	12540
8	1	128	1905	10	66656	30480
2	8	64	217	5	7240	6944
2	8	96	517	8	17908	24816
2	8	128	945	9	33312	60480
4	4	64	225	7	7632	3600
4	4	96	529	10	18496	12696
4	4	128	961	12	34096	30752
8	2	64	217	4	7048	1736
8	2	96	517	8	17620	6204
8	2	128	945	10	32928	15120
1	16	64	189	3	5676	12096
1	16	96	475	5	15562	45600
1	16	128	889	6	30184	113792
16	1	64	189	1	5196	756
16	1	96	475	4	14842	2850
16	1	128	889	5	29224	7112

4.2. Performance analysis of SC-IF-PCG-QSC.

In this section, our scope is to study the performance of SC-IF-PCG-QSC and compare it with a conventional SCM (SC-PCG-QSC [Chri90c]), in which the subdomain solves are carried out directly. The computation involved in steps 1, 2 and 4 of SC-IF-PCG-QSC, which includes the incomplete factorisation of the blocks and the ICCG iterations for the subdomains is referred to as *simple iteration*, the computation involved in step 3 of SC-IF-PCG-QSC, that is the application of PCG to C, is referred to as *nested* or *outer iteration*, while the computation involved in one ICCG iteration on one subdomain is referred to as *inner iteration*. The number n_1 of points/equations/unknowns of one subdomain is given by $n_1 = \frac{n \cdot m - n_0}{\mathrm{D}}$, where n_0 is given by (2.2). We assume for simplicity that $n = m$ then $n_1 = O(\frac{n^2}{\mathrm{D}})$. Taking in account the results of the previous section the computational complexity of SC-IF-PCG-QSC is given by

$$T_{iccg}(n, \mathrm{D}, \mathrm{d}_n) = O\left(n^2 \log^2\left(\frac{n^2}{\mathrm{D}}\right) + \right. \tag{4.1}$$

$$\left. + \left[n\left(\mathrm{d}_n + \frac{\mathrm{D}}{\mathrm{d}_n}\right)\right]^{1/2} \cdot \left\{\frac{n^2}{\mathrm{D}}\right\} \log^2\left(\frac{n^2}{\mathrm{D}}\right) \cdot \mathrm{D}\right)$$

where the first term of the sum corresponds to the outer iteration, and the term inside the curly brackets to the inner iteration. We first note that the iteration always dominates the outer iteration. By comparing (2.3) and (4.1) we find out that the SC-PCG-QSC method outperforms the SC-IF-PCG-QSC when the number of subdomains D grows linearly with *n*. When increasing the number of subdomains D linearly with n^2 we would expect to have a computational complexity of $O(n^3)$ for both methods. In our numerical experiments both methods were of the same asymptotic computational complexity, but the SC-IF-PCG-QSC was always slower than SC-PCG-QSC, We believe that this is due to the overhead of ICCG because of the sparse matrix storage scheme, and possibly the differences in the software implementation of ICCG (WATSIT) and Band-LU (LINPACK). When the number of subdomains is limited and the grid size increases very much, then we expect the SC-IF-PCG-QSC method to be more effective. Figure 4.2 plots the solution time versus the *n* for SC-PCG-QSC and SC-IF-PCG-QSC and for two different number of subdomains. Among the various domain decompositions with 2 and 4 subdomains the fastest one for every method is chosen. In the case of SC-PCG-QSC the $\mathrm{d}_n \times 1$ decomposition was by far the fastest. In the case of SC-IF-PCG-QSC the difference between the fastest decomposition and the slowest one was insignificant for the same grid size and number of subdomains; the fastest decomposition for most grid sizes was the 1×2 for 2 subdomains and the 2×2 for 4 subdomains. SC-PCG-QSC was about 5 times faster than SC-IF-PCG-QSC for small grid sizes and about 4 times faster for large grid sizes.

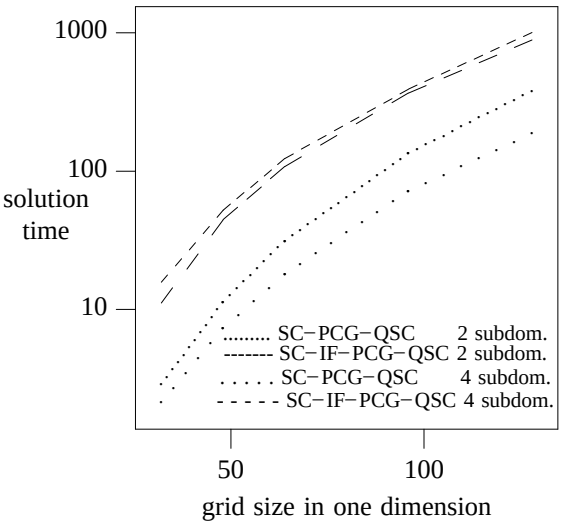

Figure 4.2. Plot of the solution time in log. scale of the Schur complement method applied to QSC equations with direct (SC-PCG-QSC) and iterative (SC-IF-PCG-QSC) solution of the subdomains versus the grid size in one dimension. Among the various domain decompositions with 2 and 4 subdomains the fastest is chosen.

Concerning memory, from the last two columns of Table 4.1 we can see that whenever the number of subdomains and the numbering of the subdomains points/equations/unknowns is such, so that the bandwidth of the blocks is kept small, then a direct solver does not waste a lot of memory, due to the limited fill-in. The memory requirements of a direct solver though are $O(n^3)$ if D is kept constant, while they are $O(n^2)$ for ICCG. Figure 4.3 plots the memory requirements versus the n for SC-PCG-QSC and SC-IF-PCG-QSC and for two different domain decompositions. We also note that the computational complexity and memory requirements of direct solvers for the subdomains are very sensitive to the numbering of the subdomain points, while this is not true for ICCG.

Summarising the above, we state that a fast iterative solver should be used for the subdomain solves, when for various reasons there is a limit in the number of subdomains, while a direct solver for the subdomain solves is more appropriate, when we can choose the domain decomposition and the numbering of the subdomain points so that the bandwidth of the blocks is kept small (approximately <16).

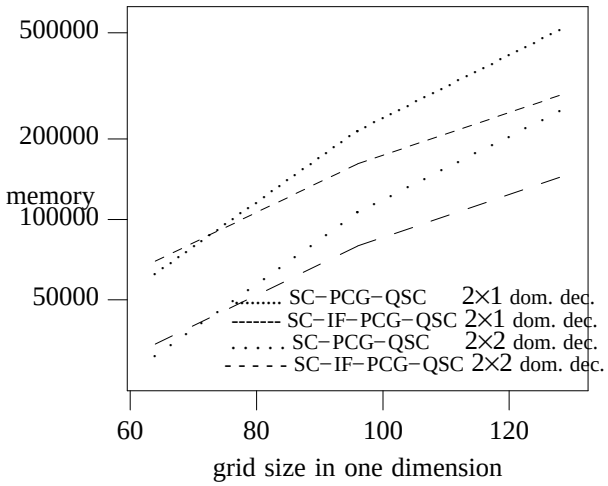

Figure 4.3. Plot of the per subdomain memory requirements in log. scale of the Schur complement method applied to QSC equations with direct (SC-PCG-QSC) and iterative (SC-IF-PCG-QSC) solution of the subdomains versus the grid size in one dimension.

5. Implementation of SC-IF-PCG-QSC on distributed memory machines and performance results on the iPSC/2 hypercube.

The implementation of SC-IF-PCG-QSC on distributed memory machines is similar to the implementation of SC-PCG-QSC, so we refer to [Chri90c] for the details of the assignment of tasks and data to processors and the communication scheme used. We point out that the application of ICCG to the subdomain blocks does not affect the communication requirements of the SCM, since the subdomain blocks are totally independent between each other. Out scope is to present some more extended computational experiments regarding the communication scheme used. For this reason, we recall that we assume a two-dimensional layout of processors either with an one-to-one mapping of subdomains to processors or with a mapping of more than one subdomain to each processor. During the application of PCG to the capacitance system a processor needs to exchange data with the (at most) four neighbour processors and with the (at most) four second nearest neighbours, due to the corner elements of the 9-point-star-like stencil of the two-step QSC method. We can accomplish though the exchange of data using only (direct) neighbour links, by following a cyclic movement of data in order. First, the data moves north, then east, then south and finally west. The product of C with a given vector requires four phases of neighbour communication, which allow no overlapping between each other, but do allow some overlapping with computation. (An alternative is to have phases 1 and 3 overlap, phases 2 and 4

overlap, but do not allow any overlapping with computation.)

We have implemented the SC-PCG-QSC method on the iPSC/2 hypercube and run experiments with up to 32 processors. The experimentally observed efficiency ranged from 99% for 2 processors to 82% for 32 processors, assuming a fixed grid size and domain decomposition.

We tried to measure the overhead due to communication by measuring the execution time, when the processors run performing only the communication statements, including any indexing required due to the communication routines. Note that the algorithm is not fully load balanced, since there are processors that have to carry out more computation than others (for example the south-west corner processor) due to the one-sided assignment of interfaces to processors and there are processors with less communication requirements than others (for example all corner processors). With the above experiments we can only measure the overhead related to communication, including indexing and synchronisation due to communication. Other reasons for which the efficiency of our implementation degrades from 100% are the imbalance in computation and the synchronisation due to that, as well as the non-scaling of the problem size with the number of processors. In Table 5.1 we show the communication overhead of one PCG iteration applied to the capacitance system and measured according to the above for several grid sizes and layouts of processors.

Table 5.1. Communication time in milliseconds for the four-phase local exchange of data of the Schur complement method for QSC equations, for various grid sizes, domain decompositions and processor configurations on the iPSC/2.

processors	2	4	8	16		32		
layout	2×1	4×1	8×1	8×2	16×1	8×4	16×2	32×1
$n+1$								
48	1.28	3.20	3.20	2.56	3.20	2.88	2.56	
64	1.60	3.20	3.20	4.16	3.20	2.88	4.16	3.52
96	1.60	3.20	3.20	4.16	3.84	2.88	4.16	3.52

According to the iPSC/2 performance report a message of up to 100 bytes (25 numbers) takes about the same amount of time to be transferred between neighbour processors, while a message just above 100 bytes takes about double the time, due to the use of a different protocol. Table 5.2 shows the latency for the transfer of various sizes messages between neighbour processors, according to the Intel benchmarks.

There are some interesting notes to make, regarding the choice of domain decomposition and communication overhead, which are discussed below.

1. A domain decomposition (and also processor layout in this case) into 2×1 (or 1×2) subdomains has the advantage of one-pass communication, and thus minimal communication overhead.
2. The d_n×1 (or 1×d_m) processor layouts where $d_n > 2$ (or $d_m > 2$) are equivalent with respect to communication independently of $d_n = D$. They require a two-pass communication.
3. Some layouts (e.g. 8×2, 16×2) are substantially affected by the grid size n, while others are not (e.g. 4×1, 8×1, 16×1, 32×1). This happens because in some cases the size of the message, which is almost equal to the number of a subdomain's points in one direction, drops below 25. For example, in the case of 8×2 layout, the size of messages sent east-west is below 25, when the grid size is 48, while it is more than 25, when the grid size is 64.
4. The 8×4 layout is the best for 32 processors, since all messages sent (north-south and east-west) have size less than 25.

The above observations allow us to state, that when the number of processors we want to utilise is greater than 2, the domain decompositions with about equal number of subdomains in each direction should be preferred, if the ratio of the grid size over the number of subdomains in each direction is small (order of 25), otherwise (when the ratio is large) stripwise decompositions should be preferred, in order to minimise the communication overhead. Moreover, if we can choose the domain decomposition so that the number of subdomain points in each direction of a subdomain is kept below 25, then the communication is minimal.

Table 5.2. Communication time in milliseconds for several size messages between direct neighbour processors on the iPSC/2.

bytes	0	8	64	100	104	1024	8192	16384
latency	0.55	0.55	0.56	0.57	1.01	1.34	3.92	6.87
bandwidth	0.00	14.54	114.38	175.44	102.97	764.18	2089.80	234.86

6. Comparison with Schwarz splitting methods.

In this last section, we present preliminary results of a comparison between DD methods and SS methods.

SS methods assume the decomposition of the domain of problem definition into overlapping subdomains, so that each subdomain has a fictitious boundary, which belongs to the interior of the neighbour subdomains. A generic algorithm for the SS methods is found at the end of the section.

The main advantages of SS over DD are mentioned next:

1. SS allows flexibility in the choice of discretisation method and linear solver for each subdomain.
2. When an iterative solver is used for the subdomain

problems, the final solution of a subdomain at some outer iteration can be used as an initial guess for the next outer iteration. This helps, so that the total number of inner iterations is reduced. In our experiments, we found that even 1 inner iteration per subdomain solve suffices to satisfy the stopping criteria of ICCG applied to the subdomain systems.

3. For the SS method, the number of outer iterations grows little with the grid size.

4. Only local communication is required for SS and no inner products, as in the case of DD.

5. The number of outer iterations for SS is not affected by the discretisation stencil. If we implement DD for a larger stencil discretisation method, the number of interface points will increase, thus leading to an increase in the number of iterations.

6. The amount of communication for SS is not affected by the discretisation stencil. If we implement DD for a larger stencil discretisation method, the processors need to exchange more data during the multiplication of the capacitance matrix with a given vector. For example, in the case of a stencil with 5 points in each direction, the amount of data transfer per processor and per send will be about double, compared to the one for a stencil with 3 points in each direction, like the QSC stencil.

7. With regard to load balancing of computation, the DD method has the disadvantage of not being perfectly load balanced due to the one-sided assignment of interfaces to processors. The SS method for appropriate choice of grid size and domain splitting can be more load balanced. Both methods can be load imbalanced if an iterative solver is used for the subdomain solves and different number of inner iterations are required for convergence in each subdomain solve.

The main disadvantages of SS over DD are mentioned below:

1. For the SS method, the number of outer iterations grows linearly with the number of subdomains.

2. 'Duplicate' computation is done due to overlapping in the case of the SS method.

In general, from the preliminary experiments conducted so far, SS methods have been found less efficient than DD methods. A more detailed work on this topic is under way.

Schwarz splitting algorithm:

```
choose initial fictitious boundary conditions
repeat until satisfied
    for each subdomain
        solve subdomain
            using current fictitious boundary conditions
        update fictitious boundary conditions
            using neighbours' solutions
    endfor
endrepeat
```

7. References.

[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.

[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, pp. 375–384.

[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., pp. 1267–1273.

[Chri90a] **Christara, C. C.,** *Quadratic Spline Collocation Methods for Elliptic Partial Differential Equations,* University of Toronto, DCS Tech. Rep. 235–90, March 1990, 34 pp., submitted for publication.

[Chri90b] **Christara, C. C.,** *Conjugate Gradient Methods for Spline Collocation Equations,* Proceedings of the fifth Distributed Memory Computing Conference (DMCC5), April 1990, Charleston, SC, U.S.A., pp. 550–558.

[Chri90c] **Christara, C. C.,** *Schur complement preconditioned conjugate gradient methods for spline collocation equations,* Proceedings of the 1990 International Conference on Supercomputing (ICS90), June 1990, Amsterdam, the Netherlands, pp. 108–120.

[Chri91] **Christara, C. C.,** *Multicolour orderings and iterative methods for elliptic equations* Proceedings of the Supercomputing Symposium '91, June 1991, Fredericton, New Brunswick, Canada.

[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.

[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).

[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.

[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.

[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.

[Kigh89] **Kightley, J. R.** and **P. A. Forsyth** (rev. **E. F. D'Azevedo**), *MAT1 Iterative Sparse Matrix Solver: User's Guide,* University of Waterloo, 1989.

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

[Oppe88] **Oppe, T. C., W. D. Joubert** and **D. R. Kincaid,** *NSPCG User's Guide, Version 1.0,* Center for Numerical Analysis, the University of Texas at Austin, CNA-216, April 1988.

[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.

[Tang89] **Tang, Wei-Pai,** *Generalised Schwartz Splittings,* University of Waterloo, Tech. Rep. CS-89-04 (1989).

[Tang90] Discussions of the author with **Tang, Wei-Pai,** University of Waterloo, 1990−1991.

[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 would like to thank the Cornell Theory Center for letting me use their iPSC/2 hypercube for free!