Multicolour orderings and iterative methods for elliptic equations

Christina C. Christara
Department of Computer Sciences
University of Toronto
Toronto, CANADA, M5S 1A4
e-mail: ccc@cs.toronto.edu

Abstract

In this paper we study the performance and parallel implementation of iterative methods applied to linear systems arising from the discretisation of linear elliptic Boundary Value Problems (BVP) which are ordered according to multicolour orderings. We view the discretisation of a BVP as a process of generating a set of equations between stencils. These equations, when ordered according to some indexing of the stencil, result in a sparse linear system. We discuss the effect of indexing of the stencils on the sparsity pattern of the linear system, on the convergence of iterative methods applied to the system, as well as on the parallel implementation of these methods. We discuss which stencil indexings are preferable in order to minimise communication in message-passing parallel architectures.

0. Overview

In Section 1, we briefly discuss general issues on the parallel computation of linear elliptic Boundary Value Problems (BVPs). In Section 2, we recall some common BVP discretisation methods, we view these as stencil equations, we show the type of linear system arising when some common stencil indexings are adopted. In Section 3, we present experimental results on the convergence, time complexity and memory requirements of some iterative methods on such linear systems. In Section 4, we discuss issues on the parallel implementation of iterative methods applied to linear systems as above. We focus on the communication requirements and how the chosen stencil indexing can affect these. Finally, Section 5, includes some preliminary performance results from the implementation of some iterative methods on the iPSC/2 hypercube.

© Copyright by Christina C. Christara, 1991

1. Boundary value problems and parallel computation.

The parallel computation of linear elliptic Boundary Value Problems (BVPs) in more than one dimensions has attracted the interest of increasing number of researchers over the past few years. This fact is due to several reasons, including the increasing demand of applied scientists and engineers to solve such problems accurately and efficiently, the high computational cost to obtain accurate and efficient solutions, as well as the inherent high-degree of parallelism in BVPs.

Linear Elliptic BVPs are described by an elliptic Partial Differential Equation (PDE) defined in some domain Ω

$$\mathbb{L}u(x) = g(x) \quad \text{in} \quad \Omega \tag{1.1}$$

and some boundary conditions defined on the boundary $\partial\Omega$ of Ω

$$\mathbb{B}u(x) = \gamma(x) \quad \text{on} \quad \partial\Omega \tag{1.2}$$

where $\mathbb{L}$ is a linear elliptic partial differential operator, $\mathbb{B}$ is a boundary differential operator, g and γ are given functions of x (multi-dimensional) and u is the unknown function of x.

The numerical solution of BVPs can be viewed as a three-phase processing. At first the continuous problem (1.1)-(1.2) is discretised over a set of discretisation points or elements of $\overline{\Omega} \equiv \Omega \bigcup \partial\Omega$, using some discretisation method (finite difference, finite element, etc). This is called the *discretisation* phase. The result is a set of (discrete) linear equations. The number of equations depends on how fine the discretisation is, but the number of unknowns per equation is often small and independent of the number of equations. In the second phase, called *indexing*, an ordering of the equations and unknowns is adopted, thus giving to the set of equations the form of a system, with a particular sparsity pattern. In most cases the ordering of the equations and unknowns follows the ordering of the points or elements of the discretisation. The third phase, referred to as *solution*, involves the solution of the

sparse system of equations. The most time consuming phase is asymptotically the solution one, while the discretisation phase comes second in computational effort.

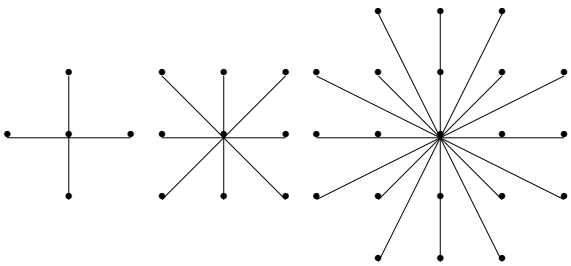
Both the discretisation and solution phases inherit high degree of parallelism. The goal is to exploit this inherent parallelism and obtain discretisation and solution modules, in which the degree of parallelism increases linearly with the size of the discrete problem, that is, with the number of equations.

The methods for the parallel computation of BVPs can be classified in three general groups: the *domain splitting* methods, in which the domain Ω of problem definition is decomposed into overlapping subdomains, the *domain decomposition* or *substructuring* methods, which are motivated by the decomposition of the domain Ω into non-overlapping subdomains, and the *matrix partitioning* methods that directly use the parallelism involved in the process of solving the linear system arising from the discretisation of the BVP problem. The ordering of equations and unknowns of the linear system plays an important role for domain decomposition methods and many matrix partitioning methods.

Matrix partitioning methods include parallel direct and parallel iterative methods. We choose to focus on parallel iterative methods for various reasons. Sequential iterative methods applied to the linear system arising from the discretisation of BVPs, often turn out to outperform sequential direct ones in both time and memory complexity. Moreover, direct methods often have a degree of parallelism growing linearly with the size of problem in one dimension. In the case of two-dimensional BVPs, this is approximately the square root of the number of equations. On the contrary, many iterative methods are easier parallelisable so that the degree of parallelism is proportional to the number of equations. In addition, communication requirements are more frequent in the parallel implementation of most direct methods, than in that of several iterative ones. We concentrate on the study of the performance and parallel implementation of some iterative methods on the linear systems arising from the discretisation of BVPs. The so called *multicolour orderings* will be the centre of our discussion. We will limit our discussion on two-dimensional second order BVPs, but all ideas expressed here can be extended to higher dimension problems. Also, for simplicity, we assume a regular (e.g. rectangular) domain Ω. From now on, by the term 'system' we will refer to the linear system arising from the discretisation of a BVP using some discretisation method.

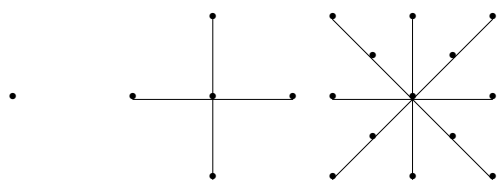
2. Discretisation stencils, dependency and multicolour orderings.

Many discretisation methods are expressed graphically through stencils, i.e. a set of (neighbouring) points appropriately positioned in the (multidimensional) space to which Ω belongs. The stencil points do not necessarily correspond to points of Ω, but to equations and/or unknowns of the system. In some cases the points are connected with lines in a form of a graph, thus showing the dependency between unknowns of the system. When appropriate coefficients are associated with every point, we get *computational molecules* [Ames77]. The number of points in the stencil of a particular discretisation method shows the number of nonzero entries in the left side of every equation obtained by applying the method to the BVP. Figure 2.1 shows the (left side) stencils of some common BVP discretisation methods. Note that such stencils can be constructed for the right side of the equations too. The most common right side stencil consists of one point, i.e. each component of the right side is computed by appropriate function evaluations on one point only. Certain discretisation methods though require function evaluations on several points, thus building a stencil of dimension higher than zero for the right side of the system. Figure 2.2 shows the right side stencils of some common BVP discretisation methods. Naturally, both the left and right side stencils shown in Figures 2.1-2.2 are for a general interior of Ω equation. Different stencils might apply on or near the boundary.



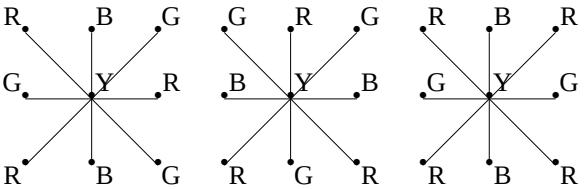
(a) (b) (c)

Figure 2.1. Some left side dependency stencils. (a) 5-point-star finite difference, (b) 9-point-star, Hodie [Lync80], quadratic spline collocation, two-step quadratic spline collocation [Chri90a], linear spline Galerkin. (c) quadratic spline Galerkin, one-step quadratic spline collocation [Chri90a].



(d) (e) (f)

Figure 2.2. Some right side dependency stencils. (d) most common methods, (e) two-step quadratic spline collocation [Chri90a], Hodie 4-th order [Lync80], (f) Hodie 6-th order [Lync80]



(a) (b) (c)

Figure 2.3. Indexing of 9-point-star stencils with four colours, red (R), black (B), green (G) and yellow (Y).

The sparsity pattern of the system depends on the (left side) stencil of the discretisation method and an ordering imposed on the equations/unknowns of the system. A very common ordering is the *natural* ordering, which for two-dimensional problems is bottom-up then left-to right or vice-versa. Some iterative methods are *asynchronous*, in the sense that the update of the solution approximation at some iteration depends only on vectors computed at previous iteration(s). Such methods are Jacobi and Conjugate Gradient (CG). Other iterative methods, which we will refer to as *synchronous*, require partial knowledge of vectors computed at the current iteration, for example require a forward (back) substitution for the lower (upper) triangular part of the matrix of the system. Such methods are SOR, SSOR and Preconditioned Conjugate Gradient (PCG) with incomplete factorisation or SSOR preconditioning. Note that in this paper the term (a)synchronous does not refer to the (in)dependency of computations between consecutive iterations, but to that during the same iteration. For the natural ordering, the dependency of the computations within an iteration step of the latter methods allows a degree of parallelism increasing linearly with the size of the problem in one dimension. Following a *multicolour* ordering [Orte88] we can break the dependency of computations within an iteration, in a way so that a high degree of parallelism is achieved. More specifically the degree of parallelism of synchronous iterative methods increases linearly with the (total) size of the problem divided by the number of colours used. For a 9-point star or similar stencils four colours are required in order to break the stencil dependency. The four colours can be arranged in more than one ways achieving the same goal of decoupling the unknowns. Figure 2.3 shows three four-colour indexings for stencils of type of Figure 2.1(b). These are the only ones, up to permutation of colours.

Indexings (a) and (b) of Figure 2.3 are the one the transpose of the other. If the problem behaves similarly in the x and y directions then it should make no difference to use either (a) or (b). Our discussion will be limited to type (a) and (c) four-colour indexings. If we order the equations/unknowns of the system corresponding to points of each colour in turn we can view the matrix of the system as a 4×4 block partitioned matrix. Figures 2.4-6 show the sparsity pattern of a matrix of a 9-point-star stencil for the natural and four-colour indexings (a) and (c). We have assumed that the ordering of points of same colour is done in some natural way.

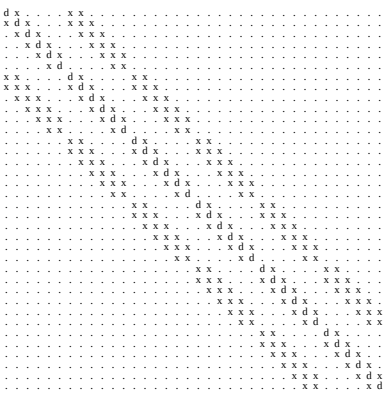


Figure 2.4. Structure of the matrix of discretisation methods with left stencil as in Figure 2.1(b), for the natural ordering. x denotes a non-zero off-diagonal element, d a non-zero diagonal one, while all zero entries are represented by character ‘.’.

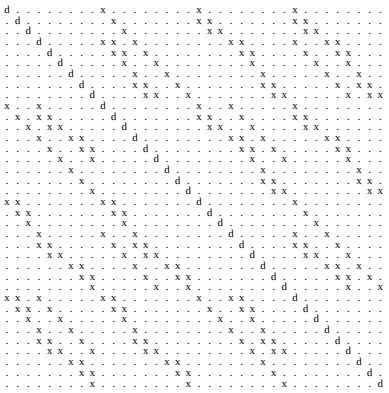


Figure 2.5. Structure of the matrix of discretisation methods with left stencil as in Figure 2.1(b), for the four-colour indexing of Figure 2.3(a). The notation of Figure 2.4 is used.

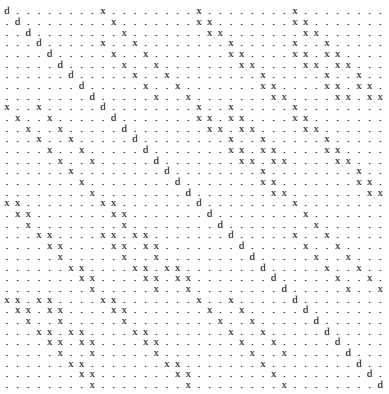


Figure 2.6. Structure of the matrix of discretisation methods with left stencil as in Figure 2.1(b), for the four-colour indexing of Figure 2.3(c). The notation of Figure 2.4 is used.

3. Performance of iterative methods on multicoloured systems.

In most cases, the ordering of equations/unknowns of a system affects the convergence of iterative methods applied to it. Unfortunately, studies pursued so far show that many iterative methods have higher rate of convergence with the natural ordering than with a multicolour one. For some iterative methods it is possible to find multicolour orderings with similar rate of convergence as for the natural [Adam86]. For others it might be not. On the other hand, as it will become clear in the next section, iterative methods applied to multicoloured systems are highly parallelisable and have few communication requests per iteration. Thus there is a trade-off between doing a lot of highly parallelisable (fast) iterations and fewer little parallelisable (slow) ones. Another trade-off is between using asynchronous methods, which are highly parallelisable and are not affected by the ordering and synchronous methods, which are usually more powerful, but also dependent on the ordering.

In the rest of this section, we present preliminary results on the performance (convergence, time complexity and memory complexity) of selected iterative methods applied to the system of two-step Quadratic Spline Collocation (QSC) equations [Chri90a]. The test problem was

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

$$u = g \quad \text{in } \partial\Omega, \tag{3.2}$$

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 various iterative methods we used the 'NSPCG' package from the Center for Numerical Analysis, at the University of Texas at Austin [Oppe88]. The stopping criterion used was the relative Euclidean norm of the residual at the current iteration, i.e. $||r^{(k)}||_2 \le \varepsilon ||b||_2$, where $r^{(k)}$ is the residual at the k-th iteration, b the right side vector and $||.||_2$ the Euclidean norm of a vector. The desired precision was chosen to be $\varepsilon = 10^{-5}$. The initial guess that was used for the first step of the QSC method was the zero vector, while in the second step we use the already computed second order solution vector from the first step. All computation was carried out in double precision on a SUN 4/60 Sparcstation. The selected iterative methods are:

RICH-CG: implements the standard CG algorithm without preconditioning.

SSOR-CG: implements the PCG algorithm with the SSOR preconditioner $(D+L)D^{-1}(D+U)$, where D, L and U are diagonal/lower triangular/upper triangular matrices holding the respective elements of the matrix of the system.

SSOR-SRCG: implements the PCG algorithm with the SSOR preconditioner $(\frac{1}{\omega}D+L)(\frac{1}{\omega}D)^{-1}(\frac{1}{\omega}D+U)$, and adaptive procedures for ω.

ICi-CG: implements the PCG algorithm with preconditioner the *level* i incomplete factor of the matrix of the system. i is a non-negative integer. The level 0 incomplete factor is the one that does not allow any fill-in. Only levels 0, 1 and 2 were tested. The user has the choice of row-sum equality modification.

All methods except RICH-CG were tested for both the natural and the multicolour ordering of

Figure 2.3(c).

In Table 3.1 we show the number of iterations required for convergence of the above iterative methods when applied to the two-step QSC equations for Problem (3.1). The ICi-CG methods results are shown without row-sum equality modification, although the ones with modification were slightly better for the test problem used. Figure 3.1 shows graphically some of the data in Table 3.1.

Table 3.1. Number of iterations required for convergence of various iterative methods applied to the two-step QSC equations for Problem (3.1), for various grid sizes. For every grid size the first row corresponds to the natural indexing and the second row to the four-colour one of Figure 2.3(c). For every iterative solver the first column corresponds to the first step of the QSC method and the second column to the second step.

grid size	RICH -CG		SSOR -CG		SSOR -SRCG		IC0 -CG		IC1 -CG		IC2 -CG	
9×9	13	12	8	6	8	6	6	5	4	4	3	3
			8	7	8	7	8	7	4	3	3	2
17×17	27	22	14	10	14	10	10	7	7	5	5	4
			16	12	19	12	15	11	7	5	5	4
33×33	54	39	26	18	19	13	17	11	12	8	8	6
			31	22	33	22	30	20	11	8	8	5
65×65	108	65	53	31	26	16	34	20	22	13	15	9
			60	37	64	37	57	36	23	14	13	8
129×129	217	129	96	57	39	19	68	34	44	22	29	14
			119	71	117	71	111	66	47	25	26	14

From Table 3.1 and Figure 3.1 we conclude that:

a) Asynchronous methods (RICH-CG) require substantially more iterations than synchronous ones (all others), with either the natural or the multicolour orderings.

b) Multicolour ordered systems require more iterations than naturally ordered ones.

c) As the level of incomplete factorisation increases the ordering becomes less significant, i.e. the benefit from using the natural ordering diminishes.

Table 3.2 shows the time taken by several iterative methods to solve the two-step QSC equations for the first step of the QSC method. The solution time includes the incomplete factorisation time required by ICi-CG methods and the iteration time. Figure 3.2 shows graphically the data of Table 3.2. We have included the solution time using standard banded Gauss elimination (Band-LU).

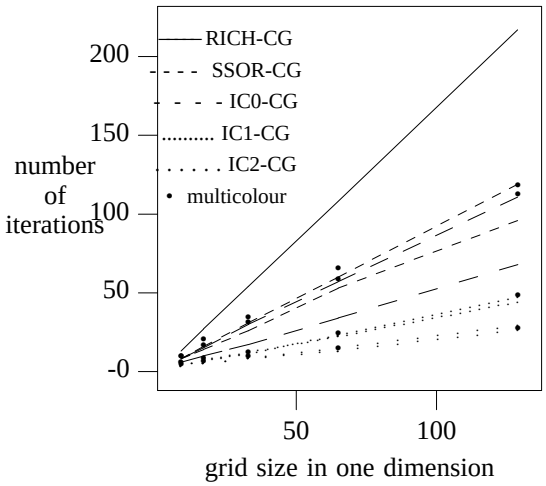


Figure 3.1. Plot of the number of iterations required for convergence of various iterative methods applied on the two-step QSC equations for the first step of the QSC method, for Problem 3.1, and for the natural and four-colour indexing of Figure 2.3(c), versus the discretisation grid size in one dimension.

Table 3.2. Time in seconds required for the solution of the two-step QSC equations by various iterative methods, for Problem (3.1) and for various grid sizes. The time for the first step of the QSC method is shown, and it includes the factorisation time for the ICi-CG methods. For every iterative solver except RICH-CG the first row corresponds to the natural indexing and the second row to the four-colour one of Figure 2.3(c).

grid size	RICH -CG	SSOR -CG	IC0 -CG	IC1 -CG	IC2 -CG
9×9	0.11	0.13	0.11	0.12	0.15
		0.11	0.22	0.19	0.35
17×17	0.74	0.75	0.58	0.63	0.87
		0.79	1.30	1.07	2.32
33×33	5.23	5.09	3.53	3.58	4.48
		5.58	9.58	6.48	12.89
65×65	43.09	41.72	28.62	25.19	28.71
		45.12	74.96	45.24	75.07
129×129	399.24	325.42	236.57	195.89	191.00
		381.62	632.25	344.27	437.60

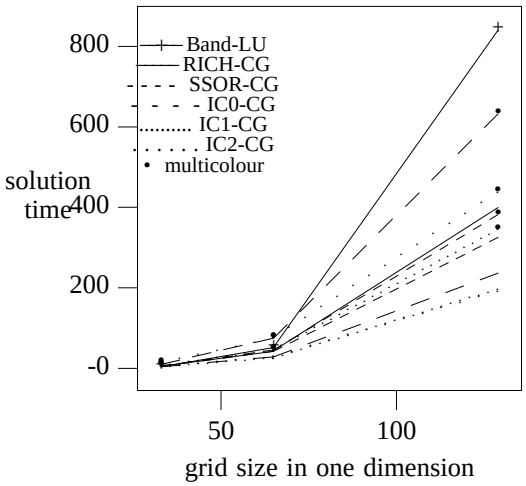


Figure 3.2. Plot of the solution time of various iterative methods applied on the two-step QSC equations for the first step of the QSC method, for Problem 3.1, and for the natural and four-colour indexing of Figure 2.3(c), versus the discretisation grid size in one dimension.

From the data of Table 3.2 and Figure 3.2 we note that as the level of incomplete factorisation increases the solution time decreases up to some point where the factorisation time starts dominating the iteration time. This point seems to be further than level 2 for natural ordered systems, while for multicoloured systems the optimal point seems to be level 1. Of course, this highly depends on the software implementation of the iterative methods. In fact, we have noticed that another software package, known as 'WATSIT', developed by [Kigh89] at the University of Waterloo, is more efficient. At this moment we base our discussion on the efficiency of the various iterative methods on the NSPCG package just for consistency among all iterative methods tested.

Another important note has to do with the memory requirements and the fill-in of the ICi-CG methods. The QSC matrix with the natural indexing has diagonals at distances 0, 1, $n-1$, n and $n+1$ from the main diagonal (speaking about the upper half of the matrix), where $n+1$ is the number of discretisation points in one dimension. The level 0 incomplete factorisation will leave the sparsity pattern unchanged, the level 1 incomplete factorisation will fill diagonals 2 and $n-2$, while the level 2 incomplete factorisation will fill diagonals 3, $n-3$ and $n-4$. This means that IC0-CG processes 5 diagonals, IC1-CG processes 7 diagonals and IC2-CG processes 10. When the system is ordered with the multicolour ordering though, the diagonals are much more scattered. In fact, the QSC matrix with the four-colour orderings has more than 10 diagonals at distances varying from 0 to almost n^2, so the incomplete factors will have more fill-in. This allows a rapid reduction of the number of iterations to the expense of memory as well as number of entries to be processed, thus making inefficient to use more than level 1 incomplete factors. Again this depends on the software implementation of the iterative methods. In Figure 3.3 we plot the memory requirements for the ICi-CG methods for i = 0, 1, 2 for both the natural and multicolour (Figure 2.3(c)) orderings. The memory requirements include integer and floating-point. We note that all methods require storage increasing linearly with the number of equations, and that the multicoloured systems require about double the memory of the naturally ordered ones.

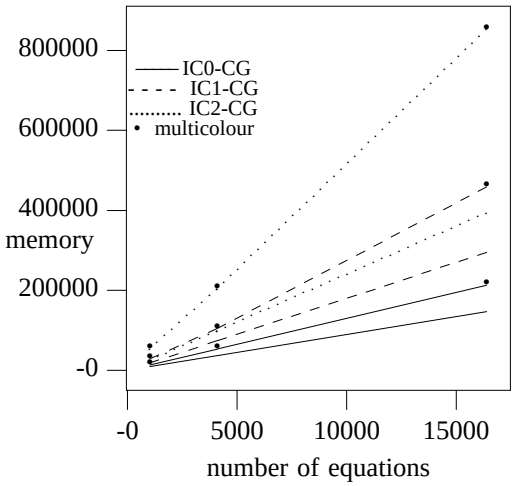


Figure 3.3. Plot of the memory requirements of incomplete factorisation methods applied on the two-step QSC equations, for the natural and four-colour indexing of Figure 2.3(c), versus the number of equations.

4. Parallel implementation of synchronous iterative methods.

In this section we discuss the mapping of the computation of synchronous iterative methods on parallel machines. The case of asynchronous methods is slightly simpler, since there is either no need for exchange of data between processors during the execution of one iteration (case of Jacobi method) or there is little exchange such as the computation of inner products and matrix-vector multiplications (case of RICH-CG discussed in [Chri90b]). In the case of synchronous iterative methods there is additional data exchange required in order to perform forward (back) substitutions for the lower (upper) part of the matrix or the incomplete factor. We will limit the discussion to multicoloured systems since this is the

case the degree of parallelism increases linearly with the size of the system.

4.1. Distribution of the data to processors.

From the construction of the multicolour orderings and the way these help to uncouple the unknowns of the system it is made clear that points/equations/unknowns with the same colour have no dependency between them. Thus it is reasonable to split same coloured points/equations/unknowns as evenly as possibly among the processors in order to achieve the highest degree of parallelism possible. Thus every processor is assigned the computation associated with some red points, some black, some green and some yellow. For simplicity, we assume that the discretisation grid is chosen appropriately so that 'some' above means one fourth of the number of equations/unknowns. The choice of which of the red (black, green, yellow) points to assign to a particular processor does not affect the degree of parallelism, but it does affect the communication requirements when a particular message-passing distributed memory architecture is chosen. At this point, we assume that we have an one-or-two-dimensional grid of processors (these can be the processors of a hypercube laid out as grid). To preserve locality of communication we assign a set of points (red, green, black and yellow) belonging to consecutive grid lines to each processor. Alternatively, we can assign a set of points belonging to a box, subset of Ω, to each processor. The rows of the matrix, the respective right side components and the respective components of any vector required during the computation are assigned similarly. All computation associated with the equations/unknowns corresponding to those points is assigned to the processor.

4.2. Parallel discretisation of the BVP problem.

The computation of the entries of the matrix assigned to a processor is carried out locally on the processor. If a discretisation method like Galerkin is used some local nearest neighbour sum up might be required. Other discretisation methods like finite difference and collocation are totally asynchronous. Thus the discretisation of a BVP in a domain Ω is carried out as a strip (line) discretisation or a box discretisation.

4.3. Exchange of data.

With the above assignment of data and computation to processors there is some need for exchange of data between processors during an iteration as well as between iterations. Depending on the iterative method chosen the exchange of data occurs during the computation of an inner product (all CG methods), during a matrix-vector multiplication (all methods), during a forward (back) substitution for the lower (upper) part of the matrix or the incomplete factor (SOR, SSOR, SSOR-CG, ICi-CG, etc), and during a triangular matrix-vector multiplication (SOR, SSOR, etc). The formation of the inner product is considered trivial when a hypercube architecture is assumed and a fan-in (preferably global) technique is used. The exchange of data in order to carry out matrix-vector multiplication is required between iterations and the procedure used is similar to that of back/forward substitutions, and triangular matrix-vector multiplications. If we assume that all the information required from the previous iteration is known to the appropriate processors, a forward substitution and an upper triangular matrix-vector multiplication requires the following:

For every colour (red, green, black, yellow) in turn do:
    Compute all its components.
    For every direct neighbour do:
        Send all components of that colour corresponding to points adjacent to the interface with that neighbour to the neighbour.
    EndFor
    For every diagonal neighbour (if there is one) do:
        Send the respective corner component of that colour to that neighbour.
    EndFor
EndFor

When the processors are arranged in an one-dimensional grid the sending of data consists of a unidirectional direct neighbour send, for every neighbour direction, assuming the order of the computation of colours is chosen appropriately. This is independent of the four-colour indexings (a) and (c) of Figure 2.3. When the processors are arranged in a two-dimensional grid the exchange is in eight directions (including the diagonal ones). In that case, the choice of four-colour indexing and of the order according to which the colours are computed can affect the communication. If the four-colour indexing of Figure 2.3(c) is adopted, at least 8 one-directional direct neighbour sends are required. If the four-colour indexing of Figure 2.3(c) is adopted, and if the computation of colours is done in the order red-green-yellow-black, 4 one-directional direct neighbour sends are required. This can be accomplished by the following algorithm:

Algorithm for forward substitution and triangular matrix-vector multiplication for the four-colour indexing of Figure 2.3(c).

1) Compute red components.

1a) Send south (new) red and (old) black components and (old) yellow corner component from south-west neighbour and (old) black corner component from west neighbour to south neighbour.

2) Compute green components.

2a) Send west (new) green and (new) red components and (old) black corner component from north-west neighbour and (new) red corner component from north neighbour to west neighbour.

3) Compute yellow components.

3a) Send north (new) yellow and (new) green components and (new) red corner component from north-east neighbour and (new) green corner component from east neighbour to north neighbour.

4) Compute black components

4a) Send east (new) black and (new) yellow components and (new) green corner component from south-east neighbour and (new) yellow corner component from south neighbour to east neighbour.

Thus, regarding communication, the four-colour indexing of Figure 2.3(c) is preferable to the other four-colour indexings. Figures 4.1 and 4.2 help visualise the type of communication required for the four-colour indexings (a) and (c) of Figure 2.3.

Summarising the communication requirements for a forward-substitution and triangular matrix-vector multiplication for four-colour indexed system of 9-point-star type we note the following: In the case of stripwise (one-dimensional) layout of processors, two sends/receives of messages of size n (or m) are required per processor. In the case of boxwise (two-dimensional) layout of processors, two sends/receives of messages of size $\frac{n}{p_n}+2$ and two sends/receives of messages of size $\frac{m}{p_m}+2$ are required per processor. In the above n and m are the sizes of problem in the x and y dimensions respectively (usually the number of grid points in one dimension plus or minus a small number, which depends on the discretisation method) and p_x and p_y are the number of processors in the x and y dimensions respectively.

When an ICi-CG method with i > 0 is used, the sparsity pattern of the incomplete factor does not allow the degree of parallelism to grow linearly with the size of the problem. As a possible solution we are experimenting [Chri91] with incomplete factors, which ignore the fill-in of the diagonal blocks (allowing fill-in of the nondiagonal ones), with a possible modification of the diagonal element.

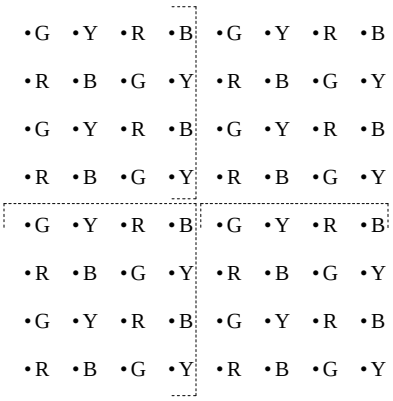


Figure 4.1. A set of 8×8 points indexed by the Figure 2.3(a) indexing, and their assignment to 4 processors in a 2×2 box partitioning.

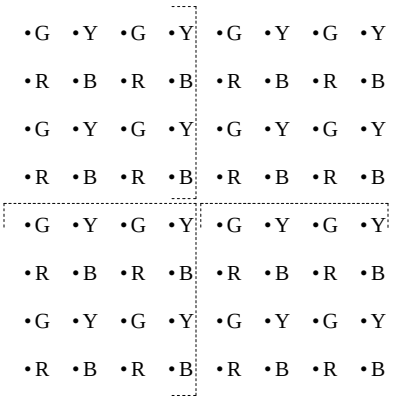


Figure 4.2. A set of 8×8 points indexed by the Figure 2.3(c) indexing, and their assignment to 4 processors in a 2×2 box partitioning.

5. Parallel performance.

We have implemented some synchronous iterative methods for multicoloured systems on the iPSC/2 hypercube and ran experiments for up to 32 processors configurations.

5.1. Communication time.

One might naturally expect that the strip assignment of points to processors will outperform the box one. This is in general true, but as our experiments show for certain problem sizes and hypercube configurations there is still some benefit in using the box assignment, because the size of the messages sent is below a threshold, under which the transfer overhead is smaller, and because the total number of data to be exchanged between processors is smaller. The communication overhead was about 7% and 6% of the total time for the 32×1 and 8×4 repectively processor

configuration, and discretisation grid size of 129×129.

5.2. Speedup and efficiency.

We measure the fixed and scaled speedup and efficiency of the methods implemented. An important note about the communication scheme is that the amount of data to be transferred per processor does not depend on the number of processors, when the problem size is scaled with the number of processors. This helps so that the degradation of the scaled speedup efficiency is minimal.

We measure the speedup of our implementation separately for the *discretisation* process, which in our experiments is the quadratic spline collocation method for the *per iteration* process, which in our experiments is one iteration of the SSOR method, and for the *solution* process, which includes some preprocessing for the startup of the SSOR iteration and the execution of all SSOR iterations. In Figure 5.1 we plot the speedup for the discretisation, per iteration and the solution processes for a fixed 129×129 discretisation grid. The estimated efficiency of the discretisation process ranges from 100% for 2 processors to 99% for 32 processors, while the efficiencies of the per iteration and solution processes range from 99% for 2 processors to 92% for 32 processors. Table 5.1 shows the data used in Figure 5.1 and some more.

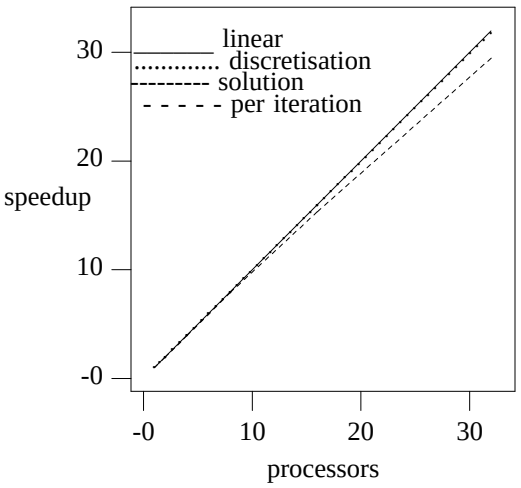


Figure 5.1. Measured speedup for the discretisation, solution and per iteration times of SSOR on the iPSC/2 for up to 32 processors configurations and fixed 129×129 grid.

Table 5.1. Time and memory performance results on the iPSC/2 for the discretisation and per iteration processes of the SSOR algorithm for various grid sizes and processor configurations. The time is in milliseconds and the memory requirements are in floating-point numbers.

grid size	processors	discretisation	per iteration	memory
65×65	1×1	10552	888.44	49152
	2×1	5277	446.04	28672
	4×1	2640	227.36	18432
	8×1	1324	118.16	13312
	8×2	672	61.44	10752
	16×1	664	63.12	10752
	8×4	332	34.02	9472
	16×2	332	33.80	9472
	32×1	332	35.92	9472
97×97	1×1	23783	1995.28	110592
	2×1	11893	995.52	64512
	4×1	5948	503.36	41472
	8×1	2976	255.52	29952
	8×2	1504	131.20	24192
	16×1	1492	132.56	24192
	8×4	747	68.90	21312
	16×2	748	68.66	21312
	32×1	748	70.88	21312
129×129	1×1	42223	4116.32	196608
	2×1	21111	1771.00	114688
	4×1	10556	889.52	73728
	8×1	5280	452.56	53248
	8×2	2648	231.00	43008
	16×1	2644	230.92	43008
	8×4	1328	117.92	37888
	16×2	1328	120.80	37888
	32×1	1328	120.32	37888

6. References.

[Adam86] **Adams, L. M.** and **H. F. Jordan,** *Is SOR color-blind?* SIAM J. Sci. Statist. Comput., **7** (1986), pp. 490–506.

[Adam88] **Adams, L. M., R. J. Levesque** and **D. M. Young,** *Analysis of the SOR iteration for the 9-point Laplacian,* SIAM J. Numer. Anal., **25** (1988), pp. 1156–1180.

[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.

[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.,** *Parallel incomplete factorisation methods,* work in progress, 1991.

[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).

[Duff89] **Duff, I. S.** and **G. A. Meurant,** *The effect of ordering on preconditioned conjugate gradients,* BIT, **29** (1989), pp. 635–657.

[Elma89] **Elman, H. C.** and **E. Agron,** *Ordering techniques for the preconditioned conjugate gradient method on parallel computers,* Computer Physics Communications, **53** (1989), pp. 237–252.

[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.

[KuoJ90] **Kuo, C.−C. Jay** and **T. Chan,** *Two-colour Fourier analysis of iterative algorithms for elliptic problems with*

red-black ordering, SIAM J. Sci. Stat. Comput., **11, 4** (1990), pp. 767–793.

[Lync80] **Lynch, R. E.** and **J. R. Rice,** *A High-Order Difference Method for Differential Equations*, Math. Comp., **34** (1980), pp. 333–372.

[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).

[Pool87] **Poole, E. L.** and **J. M. Ortega,** *Multicolor ICCG methods for vector computers*, SIAM J. Numer. Anal., **24, 6** (1987), pp. 1394–1418.

[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.

[Salt87] **Saltz, J. H., V. K. Naik** and **D. M. Nicol,** *Reduction of the effects of the communication delays in scientific algorithms on message passing MIMD architectures*, SIAM J. Sci. Statist. Comput., **8** (1987), pp. s118–s134.

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

[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!