The use of multiple fronts in Gaussian elimination [1,2]

I. S. Duff and J. A. Scott

ABSTRACT

We examine a method for extending a frontal solution scheme principally so that parallelism can be exploited in the solution process. We see also that this technique can reduce the amount of work required and enable the solution of very large problems even on uniprocessors.

Keywords: sparse matrices, frontal methods, parallel processing, PVM, domain decomposition.

AMS(MOS) subject classifications: 65F05, 65F50.

---

[1] Extended and revised version of paper in Proceedings of the Fifth SIAM Conference on Applied Linear Algebra. Edited by John Lewis. SIAM Press, 567-571.

[2] Current reports available by anonymous ftp from camelot.cc.rl.ac.uk (internet 130.246.8.61) in the directory "pub/reports". This report is in file dsRAL94040.ps.Z.

Central Computing Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

September 13, 1994.

Contents

1 Introduction

In this paper, we consider the solution of the $n \times n$ linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.1}$$

where $\mathbf{A}$ is a large sparse matrix. We consider the direct solution of (1.1) by means of sparse Gaussian elimination implemented using a frontal elimination scheme. In such a procedure, we find permutation matrices $\mathbf{P}$ and $\mathbf{Q}$, and compute the decomposition

$$\mathbf{PAQ} = \mathbf{LU}$$

where $\mathbf{L}$ is unit lower triangular and $\mathbf{U}$ is upper triangular. We consider here the case where the matrix $\mathbf{A}$ is a sum of finite-element matrices but note that the frontal scheme and the Harwell Subroutine Library (HSL) code MA42 that implements it can be used also with sparse matrices stored in standard assembled form.

We discuss frontal schemes in Section 2 and examine their strengths and weaknesses. We then indicate, in Section 3, how frontal schemes can be extended to address some of these weaknesses and discuss software designed to do so in Section 4. We examine the performance of this software on a simple test example in Section 5. Finally, we reflect on our findings and indicate possible further extensions in Section 6 and indicate the availability of the code in Section 7.

The experimental results in this paper have been obtained on a farm of DEC Alpha 3000-400 workstations and a CRAY Y-MP8I. The principal test problem we have used is an artificial finite-element problem designed to simulate those actually occurring in some CFD calculations. The elements are nine-node rectangular elements with nodes at the corners, mid-points of the sides, and centre of the element. A parameter to the element generation routine determines the number of variables per node. This parameter has been set to five for the numerical experiments in this paper. The elements are arranged in a rectangular grid whose dimensions are given in the tables.

2 Frontal schemes

The techniques we examine are based on the frontal scheme. Frontal schemes have their origins in work by Irons (1970) and the basis for our experience with them is discussed by Duff (1984). In a frontal scheme, only a submatrix of the overall sparse matrix is factorized at any one time and it can be written as

$$\begin{pmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} \\ \mathbf{F}_{21} & \mathbf{F}_{22} \end{pmatrix}, \tag{2.1}$$

where pivots can be chosen from the matrix $\mathbf{F}_{11}$ since there are no other entries in these rows and columns in the overall matrix. The rows in $\mathbf{F}_{11}$ and $\mathbf{F}_{12}$ and the columns in $\mathbf{F}_{11}$ and $\mathbf{F}_{21}$ are called *fully summed* and the block $\mathbf{F}_{11}$ is called fully summed. $\mathbf{F}_{11}$ is factorized, multipliers are stored over $\mathbf{F}_{12}$ and the Schur complement $\mathbf{F}_{22} - \mathbf{F}_{21}\mathbf{F}_{11}^{-1}\mathbf{F}_{12}$ is formed, all using full matrix kernels. At the next stage, further entries from the original matrix are assembled with this Schur complement to form another frontal matrix. The power of frontal schemes comes from the fact that they are able to solve quite large problems with modest amounts of high-speed memory and the fact that they use dense linear algebra kernels, in particular the Level 3 Basic Linear Algebra Subprograms (BLAS) (Dongarra, Du Croz, Duff, and Hammarling, 1990) for the numerical factorization.

A code for frontal elimination, called MA32, was included in the Harwell Subroutine Library in 1981 (Duff, 1981). This code was used extensively by people working with finite-elements particularly on the CRAY-2, then at Harwell. Recently we developed a significant upgrade to this package and have included it in Release 11 of HSL as MA42 (Duff and Scott, 1993). MA42 offers similar facilities to its predecessor but it makes more extensive use of higher Level BLAS than the earlier routine. On a machine with fast Level 3 BLAS, the performance can be impressive. We illustrate this point by showing the performance of our standard test problem on one processor of a CRAY Y-MP vector supercomputer, whose peak performance is 333 Mflop/s and on which the Level 3 BLAS matrix-matrix multiply routine SGEMM runs at 313 Mflop/s on sufficiently large matrices. It is important to realize that the Megaflop rates given in Table 2.1 include all overheads for the out-of-core working (see Section 4.1) used by MA42.

Table 2.1: Performance (in Mflop/s) of MA42 on CRAY Y-MP on standard test problem.

Dimension of element grid	16×16	32×32	48×48	64×64	96×96
Max order frontal matrix	195	355	515	675	995
Total order of problem	5445	21125	47045	83205	186245
Mflop/s	145	208	242	256	272

Although the Megaflop rates in Table 2.1 are very impressive, we must be very wary of getting overexcited by them. Normally the interest is to decrease computation time which could be done by reducing the number of floating-point operations. Even if the resultant Megaflop rate is lower the solution time could be decreased. We illustrate this is Table 2.2 where we use the matrix LOCK3491 from the Harwell-Boeing Collection (Duff, Grimes, and Lewis, 1992).

Table 2.2: Performance of HSL codes on CRAY Y-MP on matrix LOCK3491.

	MA42 (user order)	MA42 (MC43)	MA37 (min deg)
Mflop/s	118	89	30
Flops ($*10^6$)	1143	95	40
Time (secs)	9.7	1.1	1.3

Although MA42 solves the problem with a high Megaflop rate, the number of operations is much higher than necessary for this problem. A judicious reordering of the original problem (that is to say a reordering of the elements) can be used to reduce the operation count by decreasing the frontwidth. Algorithms for doing this are similar to those for bandwidth reduction of assembled matrices (Duff, Reid, and Scott, 1989). There is no option for doing this within MA42 but the HSL subroutine MC43 can be used to determine an ordering for subsequent calls to MA42. If this is done then, although the frontal code runs at only 89 Mflop/s, the number of floating-point operations is reduced substantially so that only 1.1 seconds are now required for the factorization compared with 9.7. It is interesting to note that, on this example, a further substantial reduction in the operation count, by using the multifrontal code MA37 with the minimum degree

ordering, does not reduce the time further. This is largely because the current version of MA37 does not use the Level 3 BLAS and the inner loop Megaflop rate is thus penalized. It should be noted that the MA37 code does produce substantially sparser factors and so subsequent solutions (forward and back-substitutions) are much quicker than for MA42. This is a graphic illustration of the risk of placing too much emphasis on computational rates (Mflop/s) when evaluating the performance of algorithms.

3 The use of multiple fronts

There are two main deficiences with the frontal solution scheme. The first, illustrated in Section 2, is that far more arithmetic is done than is required by the numerical factorization, and the second is that there is little scope for parallelism other than that which can be obtained within the higher level BLAS. One way of overcoming these problems is to extend the basic frontal algorithm to use multiple fronts. The use of multiple fronts depends upon being able to decouple the underlying "domain" and eliminate variables within each subdomain independently. The logical conclusion of this approach is the so-called multifrontal algorithm (see, for example, Duff and Reid, 1983), where many fronts are started simultaneously. The variables that are not immediately eliminated are combined to form new elements or fronts which are then merged with other new elements or original elements according to an assembly tree. The initial fronts (which may number about half the total) and the assembly tree are identified by a flop reducing ordering such as minimum degree or nested dissection. In this study we will use the term multiple fronts for algorithms for solving systems of finite-element equations which partition the finite-element domain into a number of subdomains and perform a frontal decomposition on each subdomain using an element-by-element ordering in a somewhat similar fashion to Benner, Montry, and Weigland (1987) and Zhang and Liu (1991). We reserve the term multifrontal for algorithms for solving systems of equations (which may or may not arise from a finite-element application) where a symbolic analysis to construct an assembly tree is first performed using a standard flop-reducing ordering such as minimum degree. In this section we briefly discuss the use of multiple fronts and in the next section we look at how we have been able to use the code MA42 to implement a multiple front algorithm.

With multiple fronts, we partition the finite-element domain into subdomains and perform a frontal decomposition on each subdomain separately. Since the factorizations of the subproblems are independent, this can be done in parallel. At the end of the assembly and elimination processes for the subdomains, there will remain m variables, where m is the number of variables on the interface boundaries of the subdomains. These variables are called *interface variables*. They cannot be eliminated within the subdomains since they are shared by more than one subdomain. In practice, there will also remain variables which were not eliminated within the subdomain because of stability considerations. For each subdomain, subdomain i say, the remaining Schur complement matrix for these variables can be denoted by $\mathbf{F}_i$ and the equation for the interface and uneliminated variables can be written

$$\mathbf{F}_i\mathbf{y}_i = \mathbf{c}_i \ , \tag{3.1}$$

where $\mathbf{c}_i$ is the corresponding frontal right-hand side vector (or matrix if several right-hand sides are being solved). If there are *nsub* subdomains, *nsub* equations of the type (3.1) are generated and these can be assembled to give

$$\mathbf{F}\mathbf{y} = \mathbf{c} \ , \tag{3.2}$$

where $\mathbf{F}$ is of order $m \times m$. The system (3.2) may also be solved using a frontal solver. Once (3.2) has been solved, the results for the interface variables must be passed to the

subdomains so that back-substitution can be performed. The back-substitutions on the subdomains may also be performed in parallel. Some experiments using this approach have been reported by Zhang and Liu (1991) for structural finite-element problems on an Alliant FX/80. The strategy corresponds to a bordered block diagonal ordering of the matrix. With judicious ordering within each subproblem, the overall amount of work required can be reduced; we illustrate this in Section 5.

When solving the interface problem, the order in which the subdomains are assembled is important to reduce the computational costs and storage requirements. At this stage we are essentially solving a single domain finite-element problem with a small number of elements and a good element ordering is often apparent or, if not, one may be generated using, for example, the HSL routine MC43.

As the number of subdomains increases, so does the number of interface variables m and, as a result, the cost of solving the interface problem increases and becomes a more significant part of the total computational cost. The extreme case of this is with multifrontal methods where typically over 75% of the work is performed in the top three levels of the multifrontal assembly tree (Duff, 1993, Amestoy and Duff, 1993). For the 48×48 standard test problem, with 4 subdomains the number of floating-point operations to solve the interface problem is $431 * 10^6$ and the total number of floating-point operations required is $10350 * 10^6$. If the number of subdomains is increased to 8 (all of equal size), the corresponding figures are $1295 * 10^6$ and $9065 * 10^6$, respectively. Since we are using only one processor to solve the interface problem, efficiency could be increased by nesting the multiple front algorithm. For the 8-subdomain problem, a two level tree could be formed by either combining the subdomains in pairs or in two groups of 4. On our test examples, however, we detected no advantage from doing this. Complications to the user interface make trees with more levels even less attractive. This approach was used, however, by Benner, Montry, and Weigland (1987). They partition the domain into 2^l subdomains and combine them 2 at a time so the multiple front algorithm is nested to l levels.

4 The use of MA42 for multiple front algorithms

4.1 MA42 code

Since MA42 is a complicated code, we were keen to avoid rewriting it to implement a multiple front strategy. We also wished to minimize the changes the user of MA42 would need to make in order to run MA42 in parallel. In the following subsection we describe the (minor) changes we have made to MA42. We plan to include these changes in the next release of the Harwell Subroutine Library (Release 12). We also describe three new subroutines we have written which are designed to be used in conjunction with MA42 to implement the multiple front strategy. These new subroutines comprise a new HSL package MA52.

The MA42 code can be split into three distinct phases.

(i) An prepass phase (MA42A) which accesses only the integer data and determines at which assembly each variable is fully summed. MA42A is called once by the user for each element.

(ii) A factorization phase (MA42B) which factorizes the matrix into its upper and lower triangular factors using Gaussian elimination. All calls to MA42A must be complete before the first call to MA42B is made. MA42B is called once for each element, and the calls to MA42B must be in the same order as the calls to MA42A. If

right-hand side vectors (in unassembled form) are included in the calls to MA42B, forward substitution on these right-hand sides is performed at the same time as the matrix factorization and, once the factorization is complete, MA42B calls an internal subroutine (MA42D) to perform the back-substitution.

(iii) A solution phase for further right-hand sides (MA42C). This phase is optional. MA42C calls an internal subroutine (MA42E) to perform the forward substitution on the right-hand sides and then calls MA42D to perform the back-substitution and complete the solution.

A key feature of MA42 is that it can solve large problems in a relatively small amount of main memory. In general, files for the factors are not held in the main memory. During the factorization, data is put into explicitly-held buffers and, whenever these buffers are full, they are written on to disk. If the user wishes to hold the factors in files on disk, subroutine MA42P, which initializes the disk files, must be called before the factorization phase begins. For further details of the MA42 package and specification sheets (write-ups), the reader is referred to Duff and Scott (1993).

4.2 Changes to the MA42 code

We have made only minor changes to the MA42 code. The most important change to the code has been the inclusion of an option to force diagonal pivoting. At the end of the subdomain assembly and elimination processes, we want to use MA42 to solve the interface problem (3.2) as an element problem in which the element matrices are the *nsub* frontal matrices $\mathbf{F}_i$ from (3.1). MA42 requires the user to supply a list of the indices of the variables associated with each element. Therefore, to use MA42 for the interface problem (or if we wish to nest the multiple front algorithm), the row and column indices of the variables remaining in the fronts must be the same. The Release 11 version of MA42 uses off-diagonal pivoting so that the row and column indices are not necessarily the same. We can force the row and column indices to be the same by restricting the choice of pivots to diagonal entries. However, for a single domain problem, after the last element has been assembled, it may not be possible to choose any diagonal pivots while maintaining stability and at this point a switch is made to off-diagonal pivoting to complete the factorization.

The resulting changes to the MA42 user interface are an extra control parameter ICNTL(9), one more save parameter ISAVE(46), and two additional information parameters INFO(24) and INFO(25). The new control parameter ICNTL(9) has default value 0 and, in this case, off-diagonal pivoting is used (as in Release 11). If ICNTL(9) is set by the user to 1, diagonal pivoting is enforced. ISAVE(46) is used to hold a copy of ICNTL(9) so that, if MA52 is called to implement the multiple front strategy, a check can be made that MA42B was actually employed with diagonal pivoting switched on. On exit from the final call to MA42B, INFO(24) holds the number of diagonal pivots used and INFO(25) is set to the number of off-diagonal pivots.

Another change that is necessary for a distributed memory environment is to permit the user to name the disk files for the factors. This avoids the problem of the limitation on the number of stream numbers and, in our experiments using the DEC Alpha farm, allowed us to write to local disk (see Section 5.2). A character array FILNAM of length 3, with each element of the array of length 30, and a logical variable NAME have been added to the argument list for MA42P. If the user wishes to name the disk files for factors explicitly, NAME should be set to .TRUE. and the names put in FILNAM. If NAME is .FALSE., FILNAM is not accessed and the names of the files used for the factors will depend on the computing system.

The only other change we have made to the MA42 code (which does not affect the user interface) is to use ISAVE(38) to hold a copy of the error flag INFO(1) on each exit from MA42B. This is to ensure that, if MA42B has returned an error or a warning and the user attempts to continue the multiple front computation by calling the new subroutine MA52B (see Section 4.3), the calculation will be terminated. In the Release 11 version of MA42 the parameter ISAVE(38) was unused.

4.3 The use of guard elements

We must prevent the elimination of interface variables by the factorization phase (MA42B), when applying MA42 to a subdomain. To do this, during the prepass phase, the interface variables must be marked as being not fully summed after the entry of all the elements in the subdomain. We can do this quite simply by making a final call to MA42A with an extra element containing all the interface variables. The extra element will be termed the *guard element* and there will be one guard element associated with each subdomain. After the extra call to MA42A, MA42B is called for each element in the subdomain but not for the guard element. Since MA42B is not called for the guard element, variables in the guard element (that is, the interface variables) will remain in the front. We remark that, because the number of calls to MA42B is one less than expected, the information returned by MA42B in the arrays INFO and RINFO will be incomplete.

We have provided a subroutine, MA52A, which may be used to generate the guard elements. In common with the routines in the MA42 package, MA52A uses reverse communication. The user must call MA52A once for each element in the finite-element domain. The elements can be entered in any order. In the call, the user must specify the global indices of the variables in the element and the index of the subdomain to which the element belongs. On the first entry to MA52, an integer work array IW, which must be of length at least as large as the largest integer used to index a variable in the finite-element domain, is initialized to zero. An integer array NGUARD of length equal to the number of subdomains is also initialized to zero. When an element is entered, each of its variables j is considered in turn. If IW(j) is equal to 0, variable j is being encountered for the first time and IW(j) is set to $idomn$, the index of the subdomain to which the current element belongs. If IW(j) is nonzero, say IW(j) = $jdomn$, then variable j has already been entered in subdomain $jdomn$ and so must lie on the interface between subdomains $idomn$ and $jdomn$. In this case, NGUARD($idomn$) and NGUARD($jdomn$) are incremented by one, and j is added to the lists of interface variables for subdomains $idomn$ and $jdomn$. Output from MA52A is held in an array IGUARD, whose first dimension is the number of subdomains. IGUARD($idomn$,k), k=1, ..., NGUARD($idomn$) holds the indices of the interface variables in domain $idomn$. Duplicate entries in these interface lists are removed before the return from the final call to MA52A.

4.4 Preserving the partial factorization

When the last element is passed to MA42B, it is assembled into the frontal matrix and the fully summed variables are eliminated if they pass a stability test. Since MA42B is here being used on a subdomain, the interface variables remain in the front after the assembly and eliminations for the final element. The frontal matrix and corresponding frontal right-hand side vectors are held within the work arrays used by MA42B. A new subroutine was needed to extract the frontal matrix and right-hand sides from these work arrays and return them to the user in the form of an element matrix and element right-hand side so that they can later be used in the interface problem. This new subroutine also has to write any remaining data in the in-core buffers to the files holding the factors.

We have written MA52B to perform these tasks. This subroutine can be used at any point during the sequence of calls to MA42B to preserve the partial factorization of the matrix and so can be used as a restart facility for MA42. For the multiple front algorithm, however, MA52B will only be called once for each subdomain, after the final element in the subdomain has been entered.

4.5 Interface problem

Once the forward elimination on the subdomains has been completed, we are left with the problem consisting of the interface variables and any variables that were not eliminated because of stability reasons. The system is of the form (3.2) with contributions of the form (3.1) from each subdomain. At this point, we optionally order the elements using MC43 and then solve problem (3.2) using a series of calls to MA42 in which we allow off-diagonal pivoting. There is really no problem in doing this although we must avoid overwriting any previously computed factors with those generated from the interface problem. When an element is passed to MA42B, it is assembled into the frontal matrix and the fully summed variables are eliminated if they pass a stability test. Since this is a single domain problem, when the last element is passed to MA42B, all the variables are fully summed at this stage and can be eliminated. MA42B then writes the remaining contents of the in-core buffers to the files being used to hold the matrix factors and, if right-hand sides have been entered, the solution vector is initialized to zero before, finally, the internal subroutine MA42D is called by MA42B to perform back-substitution and complete the solution of the interface problem.

4.6 Forward and back-substitution on a subdomain

Once the interface problem has been solved, we need a routine which will take the results for the interface variables and perform back-substitution on the subdomains. When designing the MA42 code, we decided that the routines to perform forward and back-substitution, namely MA42D and MA42E, should be internal subroutines. This simplified the user interface since it reduced the number of subroutine calls the user had to make. However, for the multiple front algorithm we want to be able to call forward and back-substitution directly. The routine we have written to do this is MA52C. This routine provides an interface to MA42D and MA42E. A user-supplied parameter determines whether forward or back-substitution is performed and, for back-substitution, whether it follows a call to MA42B or to MA42C for the interface problem. MA52C must be called for each subdomain but the calls may be made in parallel.

5 Numerical experiments

5.1 Performance on a uniprocessor

Although the main motivation for this work is exploitation of parallelism, it is also interesting that the amount of work can be significantly changed by partitioning the domain, and in some cases can be much reduced. For example, suppose we have an $2N \times 2N$ grid of elements where each node represents k variables. The resulting matrix is of order $(4N+1)^2k$ and bandwidth $(4N+6)k$ if ordered pagewise. Then straightforward solution requires $32k^3N^4+O(N^3)$ floating-point operations. If the domain is cut in two, the resulting operation count remains the same, but if the domain is split into four subdomains each of size $N \times N$, the operation count reduces to $18.6k^3N^4 + O(N^3)$. Note that the ordering within each subdomain is important in achieving this performance.

We illustrate this behaviour in Table 5.3 where we show the number of floating-point operations and CPU times for the same problem partitioned into differing numbers of (equal-sized) subdomains on one processor of a CRAY Y-MP.

Table 5.3: Performance of MA42 on single processor of CRAY Y-MP on 48×48 grid (47045 variables).

Number subdomains	Floating-point operations ($*10^6$)	CPU time (secs)
1	16970	69.4
2	16970	73.3
4	10350	48.4
8	9065	41.5

5.2 Performance on multiple processors

We examine the performance of our multiple front approach in two parallel environments: on an eight processor shared-memory CRAY Y-MP8I and on a network of five DEC Alpha workstations using PVM (Geist, Beguelin, Dongarra, Jiang, Manchek, and Sunderam, 1993). In each case we divide the original problem into a number of subdomains equal to the number of processors being used. It is difficult to get meaningful times in either environment because we cannot guarantee to have a dedicated machine. The times in Table 5.4 are, however, for lightly loaded machines.

Table 5.4: Multiprocessor performance of MA42 on CRAY Y-MP and DEC Alpha farm on standard test problem.

			CRAY Y-MP		DEC Alpha farm	
Dimension of element grid	Order of problem	Number of subdomains	Wall clock time (secs)	Speedup	Wall clock time (secs)	Speedup
16 x 16	5445	1	1.88	-	23.2	-
		2	1.12	1.7	29.4	0.8
		4	0.77	2.4	29.9	0.8
		8	0.92	2.0	-	-
32 x 32	21125	1	16.67	-	350.6	-
		2	9.95	1.7	250.8	1.4
		4	4.17	4.0	110.5	3.2
		8	3.97	4.2	-	-
48 x 48	47045	1	98.75	-	1460.3	-
		2	64.57	1.5	1043.0	1.4
		4	30.65	3.2	457.5	3.2
		8	15.25	6.5	-	-

The results on the CRAY are encouraging and show good speedup for large problem sizes. In general, the efficiency of the multiple front algorithm increases with the size of

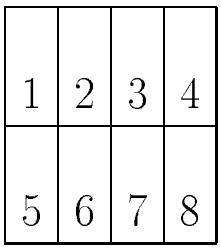
the problem. For small problems, the amount of computation on each subdomain is small and the ratio of internal variables to interface variables is also small. As a result, the communication time and time for the interface problem dominates and only a relatively small amount of work is actually performed in parallel. Results on the Alpha farm are comparable with those on the CRAY and indicate that, for larger problems, the overheads of PVM and communication costs do not dominate and good speedups are possible. We should add that it is important that disk i/o is local to each processor. The default on our Alpha system was to write all files centrally and this increased data traffic considerably, gave poor performance, and varied greatly depending on system loading. In order to avoid this problem it was necessary to explicitly name the files to enforce them to reside on the processor writing the data. This is one reason why we enhanced MA42 to accept named disk files.

We also observe that, for the larger problems, the speedup obtained when increasing the number of subdomains from 2 to 4 is greater than 2. This apparent superlinear behaviour is caused by the reduction in the number of floating-point operations for the four subdomain partitioning as discussed in Section 5.1 and illustrated in Table 5.3. We therefore ran our code with 8 subdomains on a single processor of the CRAY YMP. The times for the three grids reduced to 1.70, 12.30, and 79.85 seconds respectively, showing that the speedups are not substantially different.

5.3 Load balancing

The efficiency of a multiple front algorithm depends upon how the domain is partitioned into subdomains and how the elements within each subdomain are ordered. An important consideration when partitioning the domain into a given number of subdomains is the number of interface variables. Reducing the number of interface variables reduces the amount of data which must be transferred between processors and hence reduces communication times. It also reduces the size of the interface problem which is solved sequentially. For load balancing, each processor needs to perform a similar amount of work. For our standard test problem, if the number of required subdomains is equal to 2 or 4, it is straightforward to partition the domain and order the elements within the subdomains so that the amount of work performed within each subdomain is the same. However, if the number of subdomains is 8, it is less apparent how to subdivide the mesh to achieve load balancing.

Figure 5.1: Partition of grid in eight subdomains.



In Figure 5.1 we see that subdomains 2, 3, 6, and 7 have three internal boundaries while the remaining subdomains have only two. If the grid is a $4N \times 4N$ mesh for our standard test problem, and the subdomains are all of size $2N \times N$, those with three internal boundaries have $10N+1$ interface nodes while those with two internal boundaries

have only $6N+1$ interface nodes. These interface nodes cannot be eliminated within the subdomains and, if k is the number of freedoms per node, the subdomains with three internal boundaries have a maximum frontwidth of $(10N+5)k$ while those with two have a maximum frontwidth of $(6N+5)k$ (assuming that within each subdomain the elements are ordered parallel to the side of length N, starting at the external boundary). Thus, if the number of subdomains is the same as the number of processors, subdividing the domain into 8 equal-sized subdomains does not achieve a good load balance. An improved load balance can be achieved by subdividing the mesh into subdomains of unequal size. This is illustrated in Table 5.5 for the 48×48 test problem. In this table, subdomains 2, 3, 6, and 7 are of size $s(1) \times 24$, they have $inter(1)$ interface variables, maximum and root mean-squared frontwidths of $maxfrnt(1)$ and $rmsfrnt(1)$, while the remaining subdomains are of size $s(2) \times 24$, with $inter(2)$ interface variables, maximum and root mean-squared frontwidths of $maxfrnt(2)$ and $rmsfrnt(2)$, where $s(1) + s(2) = 24$ ($s(2) \leq s(1)$). We observe that, in this example, the total flop count is a minimum when $s(1) = 2s(2)$.

Table 5.5: Statistics for different subdivisions of 48×48 grid into eight subdomains.

$size$		$inter$		$maxfrnt$		$rmsfrnt$		Flops		Total
(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	flops
								($*10^6$)		($*10^6$)
7	17	555	415	575	435	353.0	306.5	591	1183	8326
8	16	565	405	585	425	362.2	296.9	720	1045	8303
9	15	575	395	595	415	371.4	287.2	860	918	8365
12	12	605	365	625	385	399.1	258.6	1347	596	9065

6 Conclusions and future work

It is clear from the results presented in Section 5, that the use of multiple fronts can greatly enhance and extend the power of a frontal scheme.

The effect of domain partitioning can reduce operation counts and factorization and solution times as we saw in Section 5.1. Since the work in each subdomain is of higher order than the amount of data on internal boundaries, we ensure that costs for communication and the interface problem do not dominate and so obtain good speedups even on loosely coupled networks as was shown in Section 5.2. All the runs in this paper used the same number of processors as subdomains and for our model problem and uniform partitioning, it would not help to do otherwise. However, in later experiments we will use more subdomains than processors.

There are several ways in which we intend to extend the present work. The first is to develop an ordering strategy for the partitioned problem. Applying the automatic element ordering code MC43 to each subdomain in turn will not, in general, produce efficient orderings for the subdomains since MC43 does not take into account interface variables which, once they have entered the front, cannot be eliminated. A good ordering scheme is of crucial importance for more irregular grids and for heterogeneous computing. We are also investigating strategies for automatic partitioning of irregular grids and the interaction of this with orderings on the subdomains. We are using the partitioning code CHACO (Hendrickson and Leland, 1993) in these experiments. Although our initial experiments are

not encouraging, we also plan to investigate further the nesting of the algorithm outlined in Section 3.

7 Availability of the code

MA42 and MA52 are written in standard FORTRAN 77. MA42 is included in Release 11 of the Harwell Subroutine Library and we plan to include MA52 in Release 12. A version of MA42 in Fortran 90 has also been developed and may be included in the future release. Anyone interested in using the code should contact the HSL Manager: Ms L Thick, Harwell Subroutine Library, AEA Technology, Building 8.19, Harwell, Oxfordshire, OX11 0RA, England, tel (44) 235 432688, fax (44) 235 432989, or e-mail libby.thick@aea.orgn.uk, who will provide details of price and conditions of use.

Acknowledgments
We are grateful to our colleagues at the Atlas Centre for assistance with running our codes in a multiprocessor environment. We are also grateful to Andrew Cliffe of AEA Technology, Harwell, and Hon Yau and Mark Sawyer of the Edinburgh Parallel Computing Centre for helpful discussions.

References

P. R. Amestoy and I. S. Duff. Memory allocation issues in sparse multiprocessor multifrontal methods. *Int. J. of Supercomputer Applics.*, 7:64–82, 1993.

R. E. Benner, G. R. Montry, and G. G. Weigand. Concurrent multifrontal methods: shared memory, cache, and frontwidth issues. *Int Journal of Supercomputer Applications*, 1:26–44, 1987.

J. J. Dongarra, J. J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, 1990.

I. S. Duff. *MA32 - A package for solving sparse unsymmetric systems using the frontal method. AERE R11009.* HMSO, London, 1981.

I. S. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM Journal on Scientific and Statistical Computing*, 5:270–280, 1984.

I. S. Duff. Exploitation of parallelism in direct and semi-direct solution of large sparse systems. In A. E. Fincham and B. Ford, editors, *Parallel Computation*, pages 159–174, Clarendon Press, Oxford, England, 1993.

I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

I. S. Duff and J. A. Scott. MA42 - a new frontal code for solving sparse unsymmetric systems. Technical Report RAL 93-064, Rutherford Appleton Laboratory, 1993.

I. S. Duff, J.K. Reid, and J. A. Scott. The use of profile reduction algorithms with a frontal code. *Int Journal of Numerical Methods in Engineering*, 28:2555–2568, 1989.

I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report RAL 92-086, Rutherford Appleton Laboratory, 1992.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3 User's Guide and Reference Manual. Technical Report ORNL/TM-12187, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee, 1993.

B. Hendrickson and R. Leland. The CHACO User's Guide. Version 1.0. Technical Report SAND93-2339 • UC-405, Sandia National Laboratories, Albuquerque, 1993.

B. M. Irons. A frontal solution program for finite-element analysis. *Int Journal of Numerical Methods in Engineering*, 2:5–32, 1970.

W. P. Zhang and E. M. Liu. A parallel frontal solver on the Alliant FX/80. *Computers and Structures*, 38:203–215, 1991.