RAL-TR-2001-011

The design of a portable parallel frontal solver for chemical process engineering problems

by

Jennifer A. Scott*

Abstract

We report on the design and development of a parallel frontal code for the numerical solution of the large sparse highly unsymmetric linear systems of equations that arise in industrial-scale chemical process engineering. The code has been developed as routine `HSL_MP43` for the mathematical software library HSL 2000 (http://www.cse.clrc.ac.uk/Activity/HSL). The main design goals for `HSL_MP43` were: portability, ease of use, efficiency, and flexibility. We discuss how each of these objectives are addressed within `HSL_MP43` and illustrate the performance of the code using a range of large-scale problems from chemical process simulation and optimization.

Keywords: unsymmetric linear systems, frontal method, parallel processing, Fortran 90, `MPI`.

---

*j.a.scott@rl.ac.uk
Computational Science and Engineering Department, Atlas Centre, Rutherford Appleton Laboratory, Oxon OX11 0QX, England.
This work was funded by the EPSRC Grant GR/M78502.
January 25, 2001.

1 Introduction

The repeated solution of large sparse highly unsymmetric linear systems of equations is generally the most computationally expensive step in the simulation of large-scale chemical processes, often requiring in excess of 90 per cent of the total run time. One possible approach to reducing the computational time while also allowing larger problems to be solved, is to employ parallel algorithms that can be efficiently implemented on modern supercomputers. In a recent paper, Mallya, Zitney, Choudhary and Stadtherr (1997) introduced a parallel frontal solver PFAMP that was developed using the frontal code FAMP (Zitney and Stadtherr, 1993, Zitney, Brull, Lang and Zeller, 1995). Mallya *et. al.* demonstrated the potential of their parallel frontal solver for the linear systems arising in chemical process engineering and, through the use of a number of practical examples, illustrated the reduction in the wallclock times that can be achieved when solving such systems.

Although the results presented by Mallya *et. al.* are encouraging, PFAMP has a number of limitations. Firstly, the code was developed at Cray Research, Inc. and the University of Notre Dame specifically for use in the context of process simulation: the code is not generally available or designed for use on platforms other than CRAY machines. Secondly, PFAMP does not incorporate row ordering and so the efficiency of the code is dependent upon the ordering supplied by the user. This limits the class of problems on which the code works well. Furthermore, although the frontal solver FAMP uses partial pivoting, the parallel code incorporates more restrictive threshold pivoting. In this paper, we describe the design and development of a new general-purpose parallel solver that aims to overcome these problems. The code, `HSL_MP43`, has been developed for the mathematical software library HSL 2000 (HSL, 2000) and is available for use under licence (see `http://www.software.aeat.com/HSL2000` for details). `HSL_MP43` exploits the well established frontal solver `MA42` of Duff and Scott (1993), (1996*b*). It is written in standard Fortran 90 with `MPI` for message passing. This makes the code portable and allows it to be run on any platform on which `MPI` is available. The code is not restricted to solving process simulation problems: `HSL_MP43` may be used to solve any unsymmetric sparse linear system of equations that can be preordered to bordered block diagonal form. A key feature of the code is its use of the recent row ordering strategy of Scott (2000*b*). Furthermore, by appropriate preordering of the system matrix, `HSL_MP43` is able to use partial pivoting for stability and performance is enhanced through the use of block pivots.

This paper is organised as follows. In Section 2, we recall the frontal and multiple front methods. The design and development of our parallel frontal solver `HSL_MP43` is then discussed in Section 3. Numerical results are presented in Section 4. The performance of `HSL_MP43` is also compared with that of the frontal code `MA42` and the well known general-purpose sparse direct linear solver `MA48` of Duff and Reid (1996). Finally, in Section 5, we make some concluding remarks.

We end this section by introducing the test problems that we will use throughout this paper to illustrate the performance of `HSL_MP43`. The problems

are listed in Table 1.1. A † indicates the problem is included in the University of Florida Sparse Matrix Collection (Davis, 1997). The remaining problems were supplied by Mark Stadtherr of the University of Notre Dame. The *symmetry index* $s(A)$ of a matrix A is the number of matched nonzero off-diagonal entries (that is, the number of nonzero entries a_{ij}, $i \neq j$, for which a_{ji} is also nonzero, divided by the total number of off-diagonal nonzero entries). Small values of $s(A)$ indicate a matrix is far from symmetric while values close to 1 indicate an almost symmetric sparsity pattern. We see that the test problems, with the exception of the `ethylene` problems, are all highly unsymmetric.

Table 1.1: The test problems. † indicates problem taken from University of Florida Sparse Matrix Collection.

Identifier	Order	Number of entries	Symmetry index
`4cols`	11770	43668	0.0159
`10cols`	29496	109588	0.0167
`bayer01`†	57735	277774	0.0002
`bayer03`†	6747	56196	0.0031
`bayer04`†	20545	159082	0.0016
`bayer09`†	3083	21216	0.0212
`ethylene-1`	10673	80904	0.2973
`ethylene-2`	10353	78004	0.3020
`icomp`	75724	338711	0.0010
`lhr07c`†	7337	156508	0.0174
`lhr14c`†	14270	307858	0.0066
`lhr34c`†	35152	764014	0.0015
`lhr71c`†	70304	1528092	0.0016

Unless stated otherwise, the numerical results presented in this paper were computed on the SGI Origin2000 in Manchester, UK. All timings are wallclock times given in seconds and, in each case, are the minimum times over 10 runs. The MONET code of Hu, Maguire and Blake (2000) was used to preorder the matrix to singly bordered block diagonal form (see Sections 2.1 and 3.4.6).

2 The frontal and multiple front approaches

2.1 Background

We start by recalling the key features of the frontal and multiple front approaches. Our interest lies in solving the linear system

$$Ax = b, \tag{2.1}$$

where the $n \times n$ matrix A is large and sparse, and the right-hand side vector b and solution vector x are of length n. The frontal method is a variant of Gaussian elimination and involves the matrix factorization

$$A = PLUQ, \tag{2.2}$$

where P and Q are permutation matrices, and L and U are lower and upper triangular matrices, respectively. The solution is completed by performing the forward elimination

$$PLy = b, \tag{2.3}$$

followed by the back-substitution

$$UQx = y. \tag{2.4}$$

The main feature of the frontal method is that the nonzero entries of A are added (assembled) one row at a time into a small dense matrix, termed the *frontal matrix*. Once the last row with an entry in column l has been assembled, column l is said to be *fully summed*. Partial pivoting is performed on each column as it becomes fully summed. The pivot rows and columns are eliminated and an outer-product update on the remaining frontal matrix is performed. Since the frontal matrix is held as a full matrix, dense linear algebra kernels (in particular, high-level BLAS Dongarra, DuCroz, Duff and Hammarling, 1990) can be used for these updates, and it is this that allows the frontal method to perform at high Megaflop rates (see, for example, Duff and Scott, 1994*a*).

By writing the rows and columns of the matrix factors as they are generated on to disk (for example, in direct-access files), the frontal method may be implemented using only a small amount of main memory. The memory required is dependent on the size of the largest frontal matrix. The number of floating-point operations and the storage requirements for the matrix factors are also dependent on the size of the frontal matrix at each stage of the computation. Since the size of the frontal matrix increases when a variable enters the frontal matrix for the first time and decreases whenever a variable is eliminated, the order in which the rows are assembled is crucial for efficiency.

The frontal method has been used successfully for almost thirty years to solve a wide variety of problems. However, for modern computers, a major deficiency of the method lies in its lack of scope for parallelism other than that which can be obtained within the high-level BLAS. To circumvent this shortcoming, for problems arising from finite-element applications Duff and Scott (1994*a*) proposed allowing a (small) number of independent fronts in a somewhat similar fashion to Benner, Montry and Weigand (1987) and Zang and Liu (1991) (see also Zone and Keunings, 1991). Mallya et al. (1997) proposed extending this idea to non-element problems.

In the so-called multiple front approach for general unsymmetric problems, the matrix must be preordered to singly bordered block diagonal form

$$\begin{pmatrix} A_{11} & & & C_1 \\ & A_{22} & & C_2 \\ & & \ldots & . \\ & & A_{NN} & C_N \end{pmatrix}, \tag{2.5}$$

where the rectangular diagonal blocks A_{ll} are $m_l \times n_l$ matrices with $m_l \geq n_l$, and the border blocks C_l are $m_l \times k$. If $k_l \leq k$ is the number of columns of C_l

with at least one nonzero entry, the ordering needs to be chosen so that k is as small as possible and $k_l \ll n_l$. Ideally, the blocks should also be of a similar size. A partial LU decomposition is performed on each of the matrices

$$\begin{pmatrix} A_{ll} & C_l \end{pmatrix} \tag{2.6}$$

using the frontal method. This can be done in parallel. As the rows of (2.6) are assembled, n_l variables become fully summed and may be eliminated. These variables correspond to the columns of A_{ll}; the k_l columns of C_l do not become fully summed because they have entries in at least one other border block C_j $(j \neq l)$. Because the A_{ll} are, in general, rectangular, at the end of the assembly and elimination operations, for each block there will remain a frontal matrix F_l of order $(m_l - n_l) \times k_l$. The variables that remain in the front are termed the *interface variables* and the sum F of these remaining frontal matrices is the *interface matrix*. The $k \times k$ interface matrix may also be factorized using the frontal method. Once F has been factorized, block forward eliminations and back-substitutions can be performed (in parallel) to complete the solution.

2.2 HSL_MP43 multiple front algorithm

We now outline how the multiple front algorithm is implemented within our new multiple front code HSL_MP43. We assume the matrix has been preordered to singly bordered block diagonal form and that p processes are used $(p \leq N)$, with one process designated as the *host*. The host performs the initial checking of the data, distributes data to the remaining processes, collects computed data from the processes, solves the interface problem, and generally overseas the computation. With the other processes, the host also participates in local row ordering and in generating the partial LU decompositions.

Algorithm HSL_MP43:

Initialize: performed by host.

- Assign an equal (or near equal) number of submatrices to each of the p processes.

Reorder (parallel, optional)

- Each process generates a local row ordering for each of its assigned submatrices (A_{ll}, C_l) (see Section 2.3) and sends estimated flop counts for the frontal solver applied to the submatrix based on this local ordering to the host. We denote by P_l the permutation matrix that corresponds to the local row ordering for submatrix l.

- The host uses the flop counts to redistribute the submatrices between the processes ready for the factorization. The aim is to achieve a good load balance in terms of flops.

Frontal factorization (parallel)
For each of its assigned submatrices, process p_j performs the following steps:

- A symbolic analysis using the sparsity pattern of the matrix $\hat{A}_l$, where $\hat{A}_l$ is the $(m_l + 1) \times (n_l + k_l)$ matrix

$$\hat{A}_l = \begin{pmatrix} P_l A_{ll} & P_l C_l \\ 0 & d_l^T \end{pmatrix}, \tag{2.7}$$

  where the kth entry of the vector d_l is nonzero if and only if column k of C_l has at least one nonzero entry. The symbolic analysis determines when each column of $P_l A_{ll}$ is fully summed and computes memory requirements for the frontal elimination.

- The frontal elimination on $(P_l A_{ll}, P_l C_l)$, incorporating partial pivoting for numerical stability. Note that, because the symbolic analysis was performed on the matrix $\hat{A}_l$, which has an additional row, the columns of $P_l C_l$ do not become fully summed and in this way eliminations are restricted to the columns of $P_l A_{ll}$.

- Storage of the computed columns of L and rows of U. The rows and columns remaining in the frontal matrix are passed to the host for assembly.

Interface problem: (host).
The frontal method is used on the host to factorize the interface matrix. Note that by using the frontal method, explicitly assembly the interface matrix is avoided.

Solve (parallel).
Forward elimination on the submatrices is followed, on the host, by forward elimination and back-substitution for the interface problem. Back-substitution on the submatrices completes the computation of the solution.

The above algorithm is modified if right-hand vectors b are available at the time of the frontal factorization. In this case, forward elimination operations are performed as the partial L and U factors are generated. The frontal method then computes the solution for the interface variables and back-substitutions on the submatrices give the final solution.

`HSL_MP43` is written using the well established frontal code `MA42` (Duff and Scott, 1993). In particular, on each process subroutines from the `MA42` package are used to perform the symbolic analysis, the frontal factorization, and the forward elimination and back-substitution operations. The host also uses `MA42` to solve the interface problem.

2.3 Local row ordering

In a recent paper, Scott (2000*b*) demonstrated the importance for the performance of the multiple front algorithm of having a good local row ordering

for each of the submatrices. It is of particular importance if a number of matrices having the same sparsity pattern are to be factorized or if the factors generated are to be used repeatedly for solving for different right-hand sides b. In such instances, the effort spent on generating a good row ordering pays dividends in the resulting reductions in the overall computational times and the storage requirements. In the last few years, a number of algorithms for automatically ordering the rows of A for use with frontal solvers have been proposed (see Scott, 2000a). The most successful methods currently available are the MSRO methods of Scott (1999). Scott (2000b) proposed modifying the MSRO algorithm to take into account the columns that are not fully summed within the submatrix. A Fortran code MC62 implementing the modified MSRO algorithm has been developed for HSL 2000 and is used by HSL_MP43.

2.4 Numerical pivoting

We observe that preordering A to the form (2.5) allows the multiple front method to incorporate partial pivoting to ensure numerical stability. By contrast, the code PFAMP requires the matrix to be preordered to the form

$$\begin{pmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ldots & \\ & & & A_{NN} \\ S_1 & S_2 & & S_N \end{pmatrix}. \tag{2.8}$$

The frontal solver is applied to the diagonal block A_{ii} and the corresponding portion of the border S_i. Pivot rows can only be chosen from A_{ii} because the border rows are shared by more than one diagonal block. Thus Mallya et al. (1997) propose a partial-threshold pivoting strategy. This is not only more time-consuming but can cause pivots to be delayed, resulting in an increase in the size of the local frontal matrix beyond that predicted by the symbolic analysis phase and an increase in the size of the interface problem. These increases may lead to higher flop counts and denser factors (see Mallya et al., 1997 for further discussion).

3 The design of HSL_MP43

In this section, we look at how the new parallel frontal solver HSL_MP43 has been designed to achieve the objectives of being portable, user-friendly, efficient, and flexible. We consider each of these goals in turn.

3.1 Portability

To ensure the code can be used on a wide range of modern computers, HSL_MP43 is written in standard Fortran 90 and uses MPI for message passing. Fortran 90 was chosen not only for its efficiency for scientific computation but also because it offers many more features than Fortran 77. In particular, HSL_MP43 makes extensive use of dynamic memory allocation and this allows a much cleaner user

interface. MPI was chosen since the MPI Standard is internationally recognised and today it is widely available and accepted by users of parallel computers.

HSL_MP43 does **not** assume that there is a single file system that can be accessed by all the process. This enables the code to be used on distributed memory parallel computers as well as on shared memory machines.

One of the options offered by HSL_MP43 is for the matrix factors to be held in direct-access files. This allows much larger problems to be solved than would otherwise be possible and further increases the portability of the code by enabling it to be run on machines where each process has only a limited amount of main memory.

3.2 User interface

A key design aim for HSL_MP43 was a user interface that is straightforward and, at the same time, offers flexibility through a variety of options. Our intention was that it should be possible for the code to be used by those who have only a basic knowledge of MPI together with limited experience of Fortran 90 programming.

HSL_MP43 has a single user-callable subroutine MP43A with a single parameter data of Fortran 90 derived datatype MP43_DATA, viz

```
TYPE (MP43_DATA) ::  data
CALL MP43A (data)
```

The derived datatype has many components, only some of which are of interest to the user. A number of components must be set by the user. These include the components that define the sparsity pattern of A and the bordered block diagonal form. Other components are used by the package to provide the user with information on the computation (including frontsizes, flop counts, and factor storage). Full details of the derived datatype are provided in the user documentation.

Prior to the first call to MP43A, the user must initialize MPI by calling MPI_INIT on each process. The user must also define an MPI communicator for the package. The communicator defines the set of processes to be used. HSL_MP43 then has five separate phases:

- **Initialize:** checks user data, sets control parameters.
- **Analyse:** combines local row ordering and symbolic analysis
- **Factorize:** performs the partial LU factorization of the submatrices and factorization of the interface problem.
- **Solve:** uses the computed factors to solve for right-hand sides b.
- **Finalize** deallocates arrays and optionally deletes files holding matrix factors.

The "job" parameter data%JOB determines which phase is to be performed. Each phase must be called in order by each process, although the solve phase is optional, and the analyse and factorize phases may be combined in a single

call. If right-hand side vectors b are passed to the factorize phase, the solution x is returned to the user at the end of that phase. If the user wishes to use the matrix factors generated by the factorize phase to solve for further right-hand sides, the solve phase should be called. The user may factorize more than one matrix at the same time by running more than one instance of the package; an instance of the package is terminated by calling the finalize phase.

The symbolic analysis and factorization of the submatrices are performed using the frontal code MA42. We remark that MA42 is a reverse communication code: each time a row of the matrix being factorized is required, control is returned to the calling programme. The user of HSL_MA43 is shielded from this reverse communication: once the user has supplied the submatrix data, the computation proceeds without further action on the user's part. This arguably makes HSL_MA43 a simpler code to use and may make it the code of choice, even on a single processor machine. Performance comparisons are included in Section 4.

3.3 Efficiency

When designing HSL_MP43, particular attention was paid to making the code efficient across a range of computer platforms. As already mentioned, the frontal method is able to exploit dense linear algebra kernels. In particular, the frontal code MA42 uses Level 3 BLAS during the numerical factorization to perform the outer-product updates to the frontal matrix after pivot rows and columns have been selected. Level 3 BLAS are also used during the solve phase when solving simultaneously for multiple right-hand sides; for single right-hand sides, Level 2 BLAS are used. The use of BLAS by MA42 is explained in detail by Duff and Scott (1996*b*). Efficiency is achieved by using tailored implementations of the BLAS. In the context of finite-element applications it is often the case that a number of variables become fully summed at the same assembly step. Cliffe, Duff and Scott (1997) demonstrated that the computational time required by the frontal method may be reduced by performing more than one assembly at a time and delaying computing the outer-product updates until a block of pivots is available. Delaying updating enhances the proportion of the computation performed by Level 3 BLAS at the cost of increasing the number of flops and the number of entries in the factors. There is thus a trade-off between the use of Level 3 BLAS and the computational cost. HSL_MP43 uses a modified version of MA42 that permits the user to specify the minimum pivot blocksize; assembly of rows into the front continues until either there are no rows left to assemble or the number of fully summed columns is at least as large as the minimum pivot blocksize. In Table 3.2, factorization and solve times (for a single right-hand side) are presented for our test problems for a range of different minimum pivot blocksizes. The results are for a single processor of the Origin2000 and, in each case, the number of blocks in the singly bordered block diagonal form is 4. Based on our numerical experiments, the default value for the minimum pivot blocksize has been chosen to be 8 in HSL_MP43. We remark that Mallya and Stadtherr (1997) considered using 2×2 pivots within an early version of their multiple front code. The 2×2 pivots allowed the use of Level 3 BLAS but

Table 3.2: Timings for the factorize and solve phases for different minimum pivot blocksizes (Origin2000).

Identifier	Pivot blocksize			
	1	4	8	16
4cols	0.39/0.042	0.34/0.030	0.31/0.029	0.33/0.034
10cols	1.01/0.130	0.87/0.095	0.83/0.091	0.89/0.101
bayer01	2.44/0.245	2.24/0.237	2.22/0.220	2.28/0.233
bayer03	0.24/0.021	0.22/0.017	0.22/0.017	0.24/0.021
bayer04	1.06/0.090	0.99/0.086	0.98/0.086	1.04/0.095
bayer09	0.10/0.008	0.09/0.006	0.09/0.007	0.10/0.007
ethyelene-1	0.52/0.042	0.49/0.037	0.46/0.037	0.48/0.038
ethyelene-2	0.78/0.050	0.68/0.055	0.62/0.043	0.62/0.044
icomp	2.03/0.235	1.73/0.186	1.58/0.178	1.59/0.196
lhr07c	0.74/0.047	0.68/0.047	0.66/0.067	0.71/0.044
lhr14c	1.55/0.104	1.35/0.095	1.37/0.090	1.41/0.096
lhr34c	5.12/0.285	4.36/0.283	4.28/0.251	4.55/0.272

Mallya and Stadtherr found the performance of their code was not enhanced. They attributed this to the fact that they had available a highly optimized assembly language Level 2 BLAS routine but had no comparable optimized Level 3 BLAS routine.

For chemical process engineering applications, the user is often interested in factorizing the matrix A and then using the factors to solve repeatedly for one right-hand side after another. Through its use of the BLAS, MA42 is most efficient when solving for multiple right-hand sides. When used for solving a single right-hand side, MA42 has been found to be relatively slow compared with other sparse solvers from HSL 2000, in particular, the solve phase of MA48 can be significantly faster than MA42 (see Duff and Scott, 1996*a*).

Table 3.3: Timings for the solve phase with and without Level 2 BLAS (Origin2000).

Identifier	BLAS 2	No BLAS 2
4cols	0.016	0.011
10cols	0.045	0.034
bayer01	0.123	0.098
bayer03	0.012	0.008
bayer04	0.043	0.030
bayer09	0.006	0.004
ethylene-1	0.016	0.011
ethylene-2	0.018	0.013
icomp	0.140	0.085
lhr07c	0.016	0.013
lhr14c	0.040	0.030
lhr34c	0.116	0.110

To allow BLAS to be used when performing the forward elimination and back-substitution operations, MA42 uses direct addressing (see Duff and Scott, 1996*b*). Provided the minimum pivot block size and number of right-hand sides are sufficiently large, the overheads of copying active components of the solution to and from small dense vectors are offset by the gains from using the BLAS. However, for small pivot blocks with a single right-hand side, we have found it is more efficient to use indirect addressing (even though the BLAS cannot then be used). This is illustrated by the results presented in Table 3.3, which are for the solve phase of HSL_MP43 for a single right-hand side with and without Level 2 BLAS. The runs were performed on 4 processors of the Origin2000, using the default minimum pivot block size. We see that, in each example, the solve phase is more efficient if Level 2 BLAS is not used. For a number of problems, not using Level 2 BLAS reduces the solve time by about one third.

3.4 Flexibility

The call to the initialize phase of HSL_MP43 assigns default parameters to the control parameters. These parameters control the action and offer the user a number of options. It is these options that make the package flexible. We now discuss the main options. Full details of all the control parameters and options are given in the user documentation for HSL_MP43.

3.4.1 Input of matrix data

By default, the submatrices (A_{ll}, C_l) are held in direct-access files and the data required by a particular process must be readable by that process. For each submatrix, the data is read row-by-row as required by the process to which it is assigned. This minimises main memory requirements and data movement between processes. Alternatively, the user may hold the submatrix data in unformatted sequential files so that the data required by a process is again read by that process. If this option is used, the data for all the rows in a submatrix is read in at once, requiring more memory but, again, movement of data between processes is minimised. Options also exist for the host to read the data for each of the submatrices from sequential files or, alternatively, the user may supply the matrix data using input arrays on the host. The latter form is useful if the host has sufficient memory and the overhead for using direct-access or sequential files is high. If the data is input onto the host, there is an added overhead of sending the appropriate data from the host to the other processes. Since the host is also involved in the block factorizations, this distribution of matrix data is carried out before the factorization commences.

3.4.2 The use of direct-access files

The user may choose whether to hold the partial factors in main memory or in direct-access files. Sequential files may also be used to store the data that remains in the local frontal matrices after the partial LU decompositions. This reduces main memory requirements further and allows larger problems than could otherwise be handled to be solved. However, the extra I/O involved can

increase the overall computational time and so we suggest holding the factors and local frontal matrices in main memory unless the problem is too large to be accommodated.

3.4.3 Use of MC62 for local row ordering

Reordering the rows of the submatrices using the HSL ordering routine MC62 is optional. Although the MSRO algorithm as implemented by MC62 has been shown by Scott (1999) to perform well on a wide range of problems (including many problems from application areas outside chemical process engineering), the user may wish to supply his or her own row order. This might be the case if, for example, the problem is known to be initially well ordered or if the user wants to test the effectiveness of an alterative reordering algorithm with the multiple front approach. Moreover, if the user has already factorized a matrix with a given sparsity pattern and wishes to factorize further matrices with the same pattern, the row ordering returned from the analyse phase for the original matrix can be reused. Note that stability is ensured because the factorize phase uses partial pivoting and, for each new matrix, the pivots are recomputed. In Table 3.4, we compare the time for this so-called "fast factorization" with that for the standard factorization that includes row ordering. The timings are for the analyse phase plus the factorize phase; in each case, the number of blocks in the singly bordered block diagonal form is 4 and 4 processors of the Origin2000 are used. We see that the savings achieved by reusing the row ordering are

Table 3.4: Timings for factor and fast factor (Origin2000).

Identifier	Factor	Fast Factor
4cols	0.25	0.14
10cols	0.63	0.34
bayer01	1.73	0.95
bayer03	0.31	0.12
bayer04	1.03	0.42
bayer09	0.15	0.05
ethylene-1	1.06	0.23
ethylene-2	1.21	0.29
icomp	1.65	0.76
lhr07c	0.86	0.38
lhr14c	1.71	0.56
lhr34c	4.96	2.14

significant and conclude that, if a user needs to factorize more than one matrix with similar sparsity patterns, it is worthwhile finding a good ordering and then avoiding further reordering.

3.4.4 Minimum pivot blocksize

As discussed in Section 3.3, the size of the minimum pivot block influences the efficiency of the code. Based on our experiments, we have set the default pivot

blocksize to 8 but, as other values may be more efficient on different platforms and different problems, this is a parameter that the user may choose to reset.

3.4.5 Storage of L factors

If the user supplies right-hand side vectors to the factorization phase, forward elimination operations are performed as the partial L and U factors are generated. Thus it is only necessary to store the L factors if the user wishes to solve later for further right-hand sides. To minimise storage requirements, `HSL_MP43` offers the user the option of not storing the L factors.

3.4.6 Preordering of A

An early design decision was not to incorporate code for preordering of the matrix A to singly bordered block diagonal form within `HSL_MP43`. There were a number of reasons for this. Firstly, in some applications, the matrix naturally occurs in the required form. Secondly, although the recent MONET code of Hu et al. (2000) performs well on the chemical process engineering problems considered in this paper, `HSL_MP43` is designed to be a general-purpose sparse solver that may be used to solve linear systems arising from different applications. In such cases, alternative approaches to reordering may need to be considered. While reordering unsymmetric matrices to bordered form remains a subject of active research, we decided not to tie `HSL_MP43` to one particular preordering algorithm but to leave this step in the hands of the user. Note, however, that we plan shortly to include a version of the MONET within HSL 2000. This will make it straightforward for the user to employ MONET and then `HSL_MP43`.

4 Numerical results

In this section, we illustrate the performance of `HSL_MP43` and compare it with that of the frontal code `MA42` and the general sparse direct solver `MA48` (Duff and Reid, 1996). Default values are used for all control parameters. `MA42` and `MA48` do not use the bordered block diagonal form (2.5). For `MA42`, the rows of A are preordered using `MC62`. For each solver, wallclock timings (in seconds) are presented for three execution paths, namely:

1. Analyse + Factorize + Solve (AFS) : This is the time required to perform the analyse phase, to determine a pivot sequence, to compute the L and U factors of A, and to perform the forward elimination and back-substitution operations to solve $Ax = b$ for a single right-hand side b.

2. Fast Factorize (FF) : This is the time taken to factorize a matrix having the same sparsity pattern as one that has already been factorized.

3. Solve (S): This is the time to solve $Ax = b$ by performing forward elimination and back-substitution operations using previously computed L and U factors of A.

We first present timings on the Origin2000 for HSL_MP43 run on 1, 2, 4 and 8 processors, and compare it with MA42 and MA48 run on a single processor. For HSL_MP43, the number of blocks in the singly bordered block diagonal form is 8. We have not run on more than 8 processors because most of the problems are not large enough to reorder into more than 8 blocks. As the number of blocks increases, so does the number of interface variables and, for small problems, solving the interface problem can quickly dominate the overall computational cost. The times required for AFS are given in Table 4.5. The time taken to preorder the rows of A for MA42 using MC62 is included in the analyse time. In Table 4.6, timings are presented for the fast factorize FF and, in Table 4.7, timings are given for the solve S.

Table 4.5: Timings for analyse+factorize+solve (Origin2000). NS denotes not solved.

Identifier	MA42	MA48	MP43 No. processors			
			1	2	4	8
4cols	0.76	1.76	0.85	0.49	0.32	0.20
10cols	2.11	4.93	2.09	1.21	0.69	0.46
bayer01	5.47	4.45	5.14	2.90	1.91	1.16
bayer03	0.67	0.41	0.90	0.53	0.31	0.20
bayer04	2.53	1.91	3.15	1.81	1.10	0.69
bayer09	0.24	0.10	0.36	0.22	0.15	0.11
ethylene-1	6.49	0.56	1.84	1.11	0.68	0.40
ethylene-2	4.01	0.58	2.27	1.28	0.88	0.53
icomp	4.47	0.67	5.23	2.97	1.88	1.17
lhr07c	1.94	2.40	2.69	1.67	0.97	0.73
lhr14c	4.11	5.15	5.46	3.10	1.84	1.21
lhr34c	11.03	16.40	16.14	8.71	5.15	3.55
lhr71c	29.94	NS	33.47	18.27	10.37	6.78

We see that, in general, on a single processor the AFS time for MA42 is faster than that for HSL_MP43. There are a number of exceptions, notably the ethylene problems. For these problems, using the MSRO algorithm to order the rows of A for MA42 does not improve the row order sufficiently for the frontal method to perform well. For these problems, by preordering A to bordered block diagonal form and then ordering the rows within each block and using the multiple front approach, we are able to produce sparser factors. For example, for ethylene-1 the factors computed using MA42 have $35 * 10^5$ entries whereas the HSL_MP43 factors have a total of only $7.8 * 10^5$ entries. HSL_MP43 is, of course, designed to be run on more than one processor and, for each problem, using only two processors, HSL_MP43 outperforms MA42. The performance of HSL_MA43 improves as the number of processors increases to 4 and to 8, although for the smallest problems the speedups achieved are less than for the large problems. The MA42 solve is significantly slower than HSL_MP43. This is because MA42 uses the BLAS during the forward elimination and back-substitution operations and, as discussed in Section 3.3, on the Origin2000 it

was found to be faster to use indirect addressing and no BLAS when solving for a single right hand side. Currently, MA42 does not offer the user the option of not using the BLAS.

Table 4.6: Timings for fast factorize (Origin2000). NS denotes not solved.

Identifier	MA42	MA48	MP43 No. processors			
			1	2	4	8
4cols	0.40	0.24	0.38	0.23	0.18	0.12
10cols	1.18	0.55	0.93	0.56	0.34	0.26
bayer01	3.21	0.48	2.21	1.30	0.92	0.58
bayer03	0.45	0.05	0.29	0.18	0.13	0.10
bayer04	0.95	0.21	0.98	0.62	0.42	0.33
bayer09	0.06	0.02	0.11	0.08	0.07	0.06
ethylene-1	4.35	0.06	0.34	0.22	0.18	0.14
ethylene-2	3.27	0.07	0.51	0.32	0.20	0.16
icomp	1.88	0.11	1.75	1.06	0.84	0.52
lhr07c	0.38	0.44	0.84	0.61	0.47	0.42
lhr14c	1.03	0.83	1.54	0.97	0.72	0.58
lhr34c	3.30	2.62	5.35	3.18	2.23	1.80
lhr71c	11.16	NS	11.62	7.14	4.49	3.77

Table 4.7: Timings for solve (Origin2000). NS denotes not solved.

Identifier	MA42	MA48	MP43 No. processors			
			1	2	4	8
4cols	0.068	0.013	0.031	0.020	0.014	0.010
10cols	0.194	0.047	0.088	0.078	0.036	0.027
bayer01	0.421	0.083	0.214	0.188	0.117	0.067
bayer03	0.192	0.005	0.021	0.011	0.008	0.007
bayer04	0.142	0.027	0.083	0.053	0.030	0.023
bayer09	0.016	0.002	0.006	0.005	0.004	0.004
ethylene-1	0.202	0.009	0.026	0.014	0.011	0.007
ethylene-2	0.175	0.009	0.037	0.019	0.010	0.008
icomp	0.415	0.049	0.178	0.124	0.101	0.073
lhr07c	0.040	0.028	0.045	0.033	0.019	0.012
lhr14c	0.094	0.060	0.095	0.062	0.037	0.033
lhr34c	0.270	0.169	0.267	0.180	0.125	0.086
lhr71c	0.939	NS	0.563	0.463	0.235	0.171

On a single processor, HSL_MP43 generally does not perform as well as MA48. However, the analyse phase of MA48 (with the default parameter values used in our experiments) can be relatively slow so that, for some problems (including 4cols and 10cols), MA42 and HSL_MP43 on a single processor are significantly faster than MA48 for AFS. As the number of processors increases, HSL_MP43 generally outperforms MA48, particularly when used for the largest problems,

for which it is primarily designed. We note that we were not able to use MA48 to solve problem lhr71c because there was insufficient memory (denoted by NS), emphasising the limitations of traditional serial sparse solvers. Although the fast factorize for MA48 is faster than for HSL_MP43, it is more restrictive. The fast factorize phase of MA48 uses exactly the same pivot sequence that was computed on the initial factorization. It should, therefore, only be used if the user is confident that the changes to the numerical values of the matrix have not made this sequence unsuitable. To check the solution, the user may decide to use the iterative refinement option offered by the solve phase of MA48. This adds a significant overhead to the execution time, but is not included in our results (see Duff and Reid, 1996). By contrast, HSL_MP43 only reuses the row assembly order and uses partial pivoting for stability for the initial and subsequent factorizations.

4.1 Timings on other platforms

Experiments have also been performed on other platforms. In Table 4.8 we present timings for HSL_MP43 on a Compaq DS20, which has 2 processors. Desktop computers with a small number of processors are increasingly common as they become more affordable and our results illustrate the very worthwhile speedups that can be achieved using just 2 processors.

Table 4.8: Timings for HSL_MP43 run on 1 and 2 processors of a Compaq DS20.

Identifier	AFS		FF		S	
	1	2	1	2	1	2
4cols	0.31	0.17	0.17	0.09	0.019	0.011
10cols	0.81	0.44	0.43	0.24	0.061	0.039
bayer01	2.70	1.57	1.62	0.89	0.171	0.115
bayer03	0.33	0.19	0.09	0.06	0.010	0.006
bayer04	1.56	0.96	0.80	0.51	0.063	0.040
bayer09	0.12	0.04	0.03	0.04	0.004	0.002
ethylene-1	1.89	1.49	1.21	0.96	0.081	0.062
ethylene-2	0.93	0.55	0.43	0.26	0.033	0.021
icomp	2.31	1.37	1.09	0.67	0.093	0.067
lhr07c	0.93	0.54	0.28	0.17	0.028	0.017
lhr14c	2.04	1.15	0.69	0.39	0.064	0.043
lhr34c	6.26	3.68	2.68	1.71	0.173	0.114
lhr71c	16.44	9.11	8.40	4.76	0.405	0.254
Average speedup		1.72		1.72		1.64

In Table 4.9 timings are given for HSL_MP43 when run on 2, 4 and 8 processors of the Cray T3E-1200E at Manchester, UK. As in the other experiments reported on in this paper, the factors were held in main memory and no use was made of direct access files. Working in main memory, it was not possible to solve lhr71c using only 2 processors (denoted by NS). Clearly, we could use the option offered by HSL_MP43 to hold the factors in direct access files, but in our experience it is not possible to achieve consistent (repeatable) timings on the

T3E if there is a significant amount of I/O. Again, good speedups are achieved as the number of processors increases, particularly for the larger problems.

Table 4.9: Timings for HSL_MP43 run on 2, 4, and 8 processors of a Cray T3E. NS denotes not solved.

Identifier	AFS			FF			S		
	2	4	8	2	4	8	2	4	8
4cols	0.50	0.31	0.20	0.28	0.19	0.14	0.026	0.021	0.015
10cols	1.14	0.65	0.40	0.65	0.39	0.25	0.060	0.040	0.030
bayer01	2.59	1.63	0.94	1.41	0.95	0.56	0.128	0.096	0.061
bayer03	0.44	0.27	0.17	0.21	0.14	0.10	0.018	0.013	0.12
bayer04	1.56	0.96	0.64	0.82	0.53	0.40	0.054	0.036	0.28
bayer09	0.18	0.12	0.10	0.09	0.07	0.06	0.009	0.008	0.008
ethylene-1	0.74	0.45	0.27	0.23	0.17	0.12	0.022	0.019	0.013
ethylene-2	0.90	0.57	0.38	0.39	0.24	0.20	0.028	0.018	0.015
icomp	2.49	1.61	0.88	1.05	0.88	0.47	0.112	0.094	0.059
lhr07c	1.59	1.01	0.76	0.89	0.65	0.55	0.034	0.025	0.021
lhr14c	2.87	1.69	1.28	1.47	0.95	0.88	0.056	0.043	0.030
lhr34c	9.21	5.60	3.92	5.66	3.69	2.94	0.157	0.110	0.072
lhr71c	NS	12.45	8.45	NS	8.92	6.48	NS	0.200	0.145

5 Concluding remarks

We have designed and developed a general-purpose multiple front code for solving large sparse unsymmetric systems of linear equations in parallel. The code HSL_MP43, which is in Fortran 90 with MPI for message passing, has been written using our extensive knowledge and experience of frontal methods and, in particular, uses the established frontal solver MA42 combined with the row ordering algorithm of Scott (1999). Experiments have been run on a number of practical problems arising from chemical process engineering applications and we have achieved good speedups using up to 8 processors. Numerical results have also shown that, for large problems run on 4 or 8 processors, the new code can significantly outperform the serial codes MA42 and MA48. Larger test examples are needed for experiments on more than 8 processors; we would welcome being given access to such data to perform further tests.

In this paper, we have presented HSL_MP42 timings for runs performed on an Origin2000, a 2 processor Compaq machine, and a Cray T3E. The code can, however, be run on any system with a Fortran 90 compiler and MPI available. In particular, a cluster of workstations that can communicate using MPI could be used. Results reported by Duff and Scott (1994*b*) illustrate that this kind of approach can be very effective. When working in a network-based environment, it is important to consider how the matrix data is input to the code and where the matrix factors are stored. For efficiency, the amount of data movement between processes needs to be minimised. Because of this HSL_MA43 was designed with a number of different input data options. On a cluster of

workstations, the default option is recommended whereby all the element data required by a process is read by that process. In addition, if direct-access files are needed to hold the matrix factors, the user should ensure that the files are held locally. This can be achieved by appropriately setting the parameters for the direct-access file names. Full details are given in the user documentation.

The code `HSL_MP43` is available for use under licence through HSL 2000. Anyone interested in using the codes may contact the author for details (or see http://www.cse.clrl.ac.uk/Activity/HSL).

6 Acknowledgements

I would like to thank my colleague Yifan Hu for supplying a copy of his MONET code, which was used to order the matrices in singly bordered block diagonal form. I am grateful to Mark Stadtherr of the University of Notre Dame for providing me with some of the test problems. My thanks also to Iain Duff for reading and commenting on a draft of this paper.

References

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

K.A. Cliffe, I.S. Duff, and J.A. Scott. Performance issues for frontal schemes on a cache-based high performance computer. Technical Report RAL-TR-97-001, Rutherford Appleton Laboratory, 1997.

T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, **97**(23), 1997.

J.J. Dongarra, J. DuCroz, I.S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Mathematical Software*, **16**(1), 1–17, 1990.

I.S. Duff and J.K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Mathematical Software*, **22**, 187–226, 1996.

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 and J.A. Scott. The use of multiple fronts in Gaussian elimination. Technical Report RAL-94-040, Rutherford Appleton Laboratory, 1994*a*.

I.S. Duff and J.A. Scott. The use of multiple fronts in Gaussian elimination. *in* J. Lewis, ed., 'Proceedings of the Fifth SIAM Conference Applied Linear Algebra', pp. 567–571. SIAM, 1994*b*.

I.S. Duff and J.A. Scott. A comparison of frontal software with other sparse direct solvers. Technical Report RAL-TR-96-102 (Revised), Rutherford Appleton Laboratory, 1996*a*.

I.S. Duff and J.A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996*b*.

HSL. A collection of Fortran codes for large scale scientific computation, 2000. Full details from http://www.numerical.rl.ac.uk/hsl.

Y.F. Hu, K.C.F. Maguire, and R.J. Blake. A multilevel unsymmetric matrix ordering for parallel process simulation. *Computers in Chemical Engineering*, **23**, 1631–1647, 2000.

J.U. Mallya and M.A. Stadtherr. A multifrontal approach for simulating equilibrium-stage processes on supercomputers. *Ind. Eng. Chem. Res.*, **36**, 144–155, 1997.

J.U. Mallya, S.E. Zitney, S. Choudhary, and M.A. Stadtherr. A parallel block frontal solver for large scale process simulation: reordering effects. *Computers in Chemical Engineering*, **21**, S439–S444, 1997.

J.A. Scott. A new row ordering strategy for frontal solvers. *Numerical Linear Algebra with Applications*, **6**, 1–23, 1999.

J.A. Scott. Row ordering for frontal solvers in chemical process engineering. *Computers in Chemical Engineering*, **24**, 1865–1880, 2000*a*.

J.A. Scott. Two-stage ordering for unsymmetric parallel row-by-row frontal solvers. Technical Report RAL-TR-2000-030, Rutherford Appleton Laboratory, 2000*b*.

W.P. Zang and E.M. Liu. A parallel frontal solver on the Alliant. *Computers and Structures*, **38**, 202–215, 1991.

S.E. Zitney and M.A. Stadtherr. Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Computers in Chemical Engineering*, **17**, 319–338, 1993.

S.E. Zitney, L. Brull, L. Lang, and R. Zeller. Plantwide dynamic simulation on supercomputers. *AIChE Symp. Ser.*, **91**, 313–316, 1995.

O. Zone and R. Keunings. Direct solution of two-dimensional finite element equations on distributed memory parallel computers. *in* M. Durand and F. E. Dabaghi, eds, 'High Performance Computing'. Elsevier Science Publications, 1991.