On the direct solution of very large sparse linear systems using out-of-core techniques

Jennifer A. Scott (j.a.scott@rl.ac.uk)

Joint work with John Reid

Sparse systems

We are interested in solving

$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A}$ is

LARGE

s p a r s e

- Problem sizes of order $> 10^7$ not uncommon and growing larger
- Direct methods (eg $A = PLUQ$) are popular because they are robust
- But their storage requirements generally grow more rapidly than problem size

Options for large problems

Possibilities:

- Iterative method ... but preconditioner?
- Combine iterative and direct methods? Important new area.
- Buy a bigger machine and use exisiting direct solver ... but **expensive** and **inflexible**
- Parallel direct solver?
- Use an out-of-core direct solver

Options for large problems

Possibilities:

- Iterative method ... but preconditioner?
- Combine iterative and direct methods? Important new area.
- Buy a bigger machine and use exisiting direct solver ... but **expensive** and **inflexible**
- Parallel direct solver?
- Use an out-of-core direct solver

An **out-of-core solver** holds the matrix factors in **files** and may also hold the matrix data and some work arrays in files.

Note: out-of-core working becoming even more important because of more limited local memories on distributed memory machines

Out-of-core solvers

- Idea of out-of-core solvers **not** new: band and frontal solvers developed in 1970s and 1980s held matrix data and factors out-of-core.
- For example, `MA32` in HSL (superseded in 1990s by `MA42`).
- 30 years ago John Reid at Harwell developed a Cholesky out-of-core multifrontal code `TREESOLV` for element applications.

Out-of-core solvers

- Idea of out-of-core solvers **not** new: band and frontal solvers developed in 1970s and 1980s held matrix data and factors out-of-core.
- For example, `MA32` in HSL (superseded in 1990s by `MA42`).
- 30 years ago John Reid at Harwell developed a Cholesky out-of-core multifrontal code `TREESOLV` for element applications.

More recent codes include:

- `BCSEXT-LIB` (Boeing)
- `Oblio` (Dobrian and Pothen)
- `TAUCS` (Toledo and students)
- `MUMPS` currently developing out-of-core version
- Also work by Rothberg and Schreiber

HSL_MA77

Our new out-of-core solver is HSL_MA77

- HSL_MA77 is designed to solve **LARGE** sparse symmetric systems
- HSL_MA77 implements a **multifrontal algorithm**
- First release for **positive definite** problems (Cholesky $\mathbf{A} = \mathbf{L}\mathbf{L}^\mathbf{T}$); coming VERY soon is version for **symmetric indefinite problems** ($\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\mathbf{T}$)
- Matrix data, matrix factor, and the main work space (multifrontal stack) held in **files**

Aim today: to provide brief introduction to HSL_MA77 and to present some numerical results .... hope you will go away wanting to try the code

Key features of `HSL_MA77`

- Written in Fortran 95 (HSL is Fortran library)

Key features of `HSL_MA77`

- Written in Fortran 95 (HSL is Fortran library)
- Matrix **A** may be either in assembled form or a sum of element matrices

Key features of `HSL_MA77`

- Written in Fortran 95 (HSL is Fortran library)
- Matrix $\mathbf{A}$ may be either in assembled form or a sum of element matrices
- **Reverse communication interface** with input by rows or by elements

Key features of `HSL_MA77`

- Written in Fortran 95 (HSL is Fortran library)
- Matrix **A** may be either in assembled form or a sum of element matrices
- **Reverse communication interface** with input by rows or by elements
- Separate calls for each phase
  - Entering of integer and real matrix data
  - Analyse phase (set up data structures using user-supplied pivot order)
  - Factorization (compute and store factor plus optional solve)
  - Solve (any number of right-hand sides)
  - Optional restart (save data for later factorization and/or solves)

Key features of HSL_MA77

- Written in Fortran 95 (HSL is Fortran library)
- Matrix **A** may be either in assembled form or a sum of element matrices
- **Reverse communication interface** with input by rows or by elements
- Separate calls for each phase
  - Entering of integer and real matrix data
  - Analyse phase (set up data structures using user-supplied pivot order)
  - Factorization (compute and store factor plus optional solve)
  - Solve (any number of right-hand sides)
  - Optional restart (save data for later factorization and/or solves)
- Additional flexibility through user-controlled parameters (default settings minimize decisions user must make)

Key features of HSL_MA77

- Written in Fortran 95 (HSL is Fortran library)
- Matrix **A** may be either in assembled form or a sum of element matrices
- **Reverse communication interface** with input by rows or by elements
- Separate calls for each phase
  - Entering of integer and real matrix data
  - Analyse phase (set up data structures using user-supplied pivot order)
  - Factorization (compute and store factor plus optional solve)
  - Solve (any number of right-hand sides)
  - Optional restart (save data for later factorization and/or solves)
- Additional flexibility through user-controlled parameters (default settings minimize decisions user must make)
- Separate code (HSL_MA54) written to perform the (partial) factorizations of the dense frontal matrices

Input/Output in HSL_MA77

For HSL_MA77 to perform well, the I/O **must** be efficient. I/O involves:

- writing the original real and integer data
- analyse phase (integer data only)
  - reading data for input matrix
  - writing data at each node of the assembly tree
  - reading data at each node
  - writing reordered data ready for factorization
- factorization phase
  - reading integer data at each node of the tree
  - reading real data for each leaf node
  - writing columns of L as they are computed
  - writing Schur complements to stack
  - reading matrix from stack
- solve phase
  - reading integer/ real factor data once for forward sub. and once for back sub.

Input/Output in Fortran

In Fortran 77/90/95 - standard I/O is entirely **record based**

- Fine if every read/write is of the same amount of data
- **But** we need to read/write different numbers of reals and integers at each stage of the computation
- Also, we do not want to be restricted to only accessing the data in the same order as it was written

Input/Output in Fortran

In Fortran 77/90/95 - standard I/O is entirely **record based**

- Fine if every read/write is of the same amount of data
- **But** we need to read/write different numbers of reals and integers at each stage of the computation
- Also, we do not want to be restricted to only accessing the data in the same order as it was written

We have got around these limitations while adhering to the strict Fortran standard by writing our own **virtual memory management system**

Virtual memory management

We have a separate Fortran 95 package `HSL_OF01` that handles all i/o

- `HSL_OF01` provides read/write facilities for one or more direct access files through a single **in-core buffer** (work array)
- Version for real data and another for integer data. Each has its own buffer.
- The buffer is divided into fixed length pages ... a page is the same length as a record in the file
- Careful handling of the buffer within `HSL_OF01` avoids actual input-output operations whenever possible

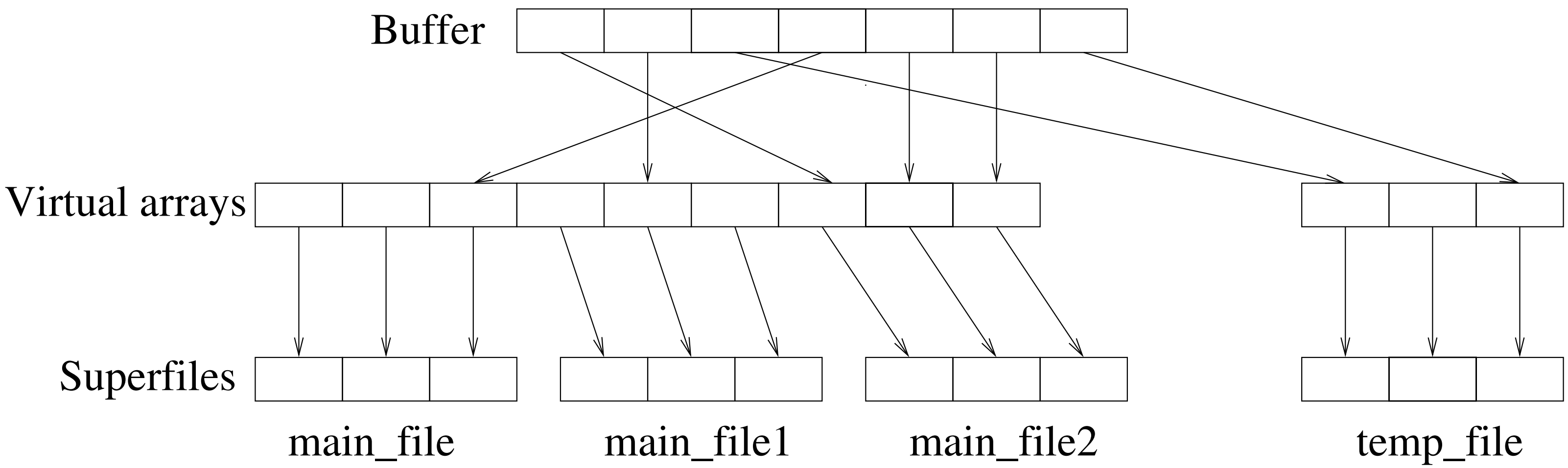
Virtual memory management

Each set of data (such as the reals in the matrix and its factor) is accessed as a **virtual array** i.e. as if it were a very long array

- Long integers (64-bit) are used for addresses in the virtual array
- Any contiguous section of the virtual array may be read or written
- Each virtual array is associated with a **primary file**
- For very large problems, the virtual array may be too large for a single file so **secondary files** are used

The primary and secondary files are **direct access files**.

Virtual memory management



- In this example, two superfiles associated with the in-core buffer
- First superfile has two secondaries, the second has none
- **Important:** user shielded from this but can control where the files are stored (primary and secondary files may be on different devices).
- Actual i/o is **not** needed if user has supplied long buffer

Use of HSL_OF01 within HSL_MA77

- HSL_MA77 has one **integer** buffer and one **real** buffer
- The integer buffer is associated with a file that holds the integer data for the matrix A and the matrix factor
- The real buffer is associated with two files:
  - one holds the real data for the matrix A and the matrix factor
  - the other is used for the multifrontal stack (work space)
- The indefinite case also uses a further real file to hold data associated with delayed pivots
- The user supplies pathnames together with names for the primary files

Use of HSL_OF01 within HSL_MA77

- HSL_MA77 has one **integer** buffer and one **real** buffer
- The integer buffer is associated with a file that holds the integer data for the matrix A and the matrix factor
- The real buffer is associated with two files:
  - one holds the real data for the matrix A and the matrix factor
  - the other is used for the multifrontal stack (work space)
- The indefinite case also uses a further real file to hold data associated with delayed pivots
- The user supplies pathnames together with names for the primary files

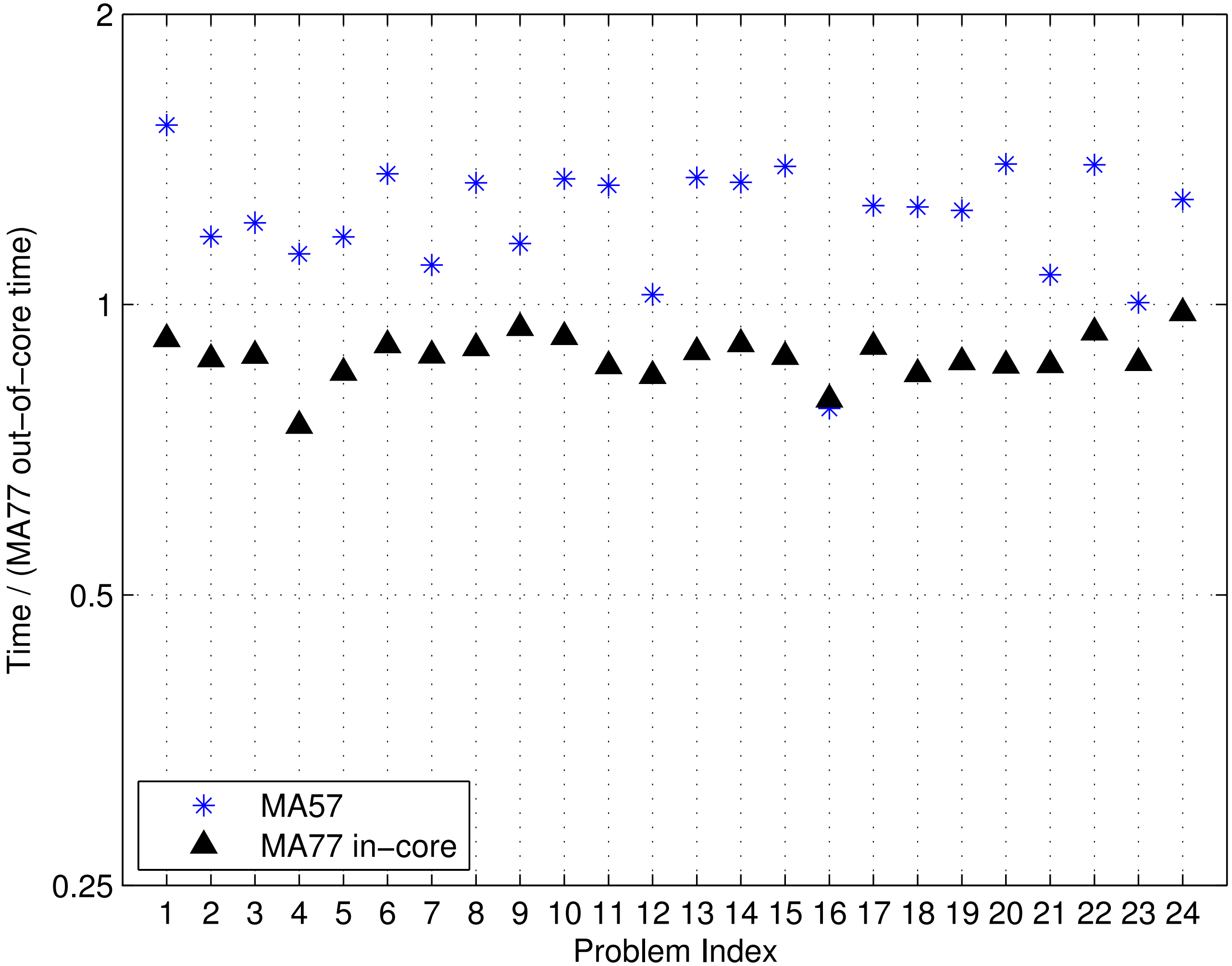
NOTE: HSL_MA77 includes option for the files to be replaced by **in-core arrays** (faster for problems for which user has enough memory) . A combination of files and arrays may be used.

Comparisons with MA57

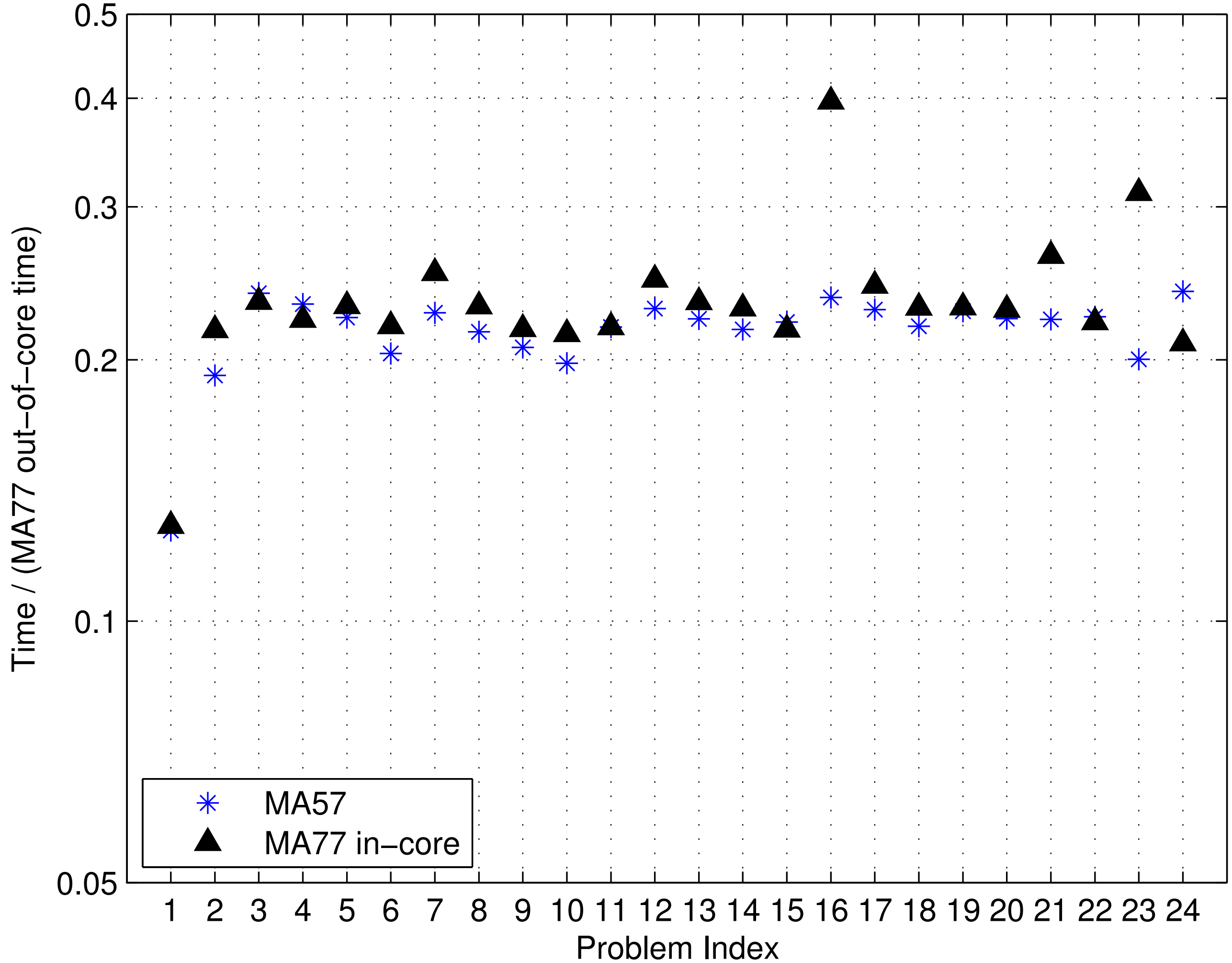
Want to compare the performance of HSL_MA77 with existing solvers.

- Test set of 24 problems of order up to $1.5 * 10^6$ from a range of applications
- All available in University of Florida Sparse Matrix Collection
- Tests used double precision (64-bit) reals on a single 3.6 GHz Intel Xeon processor of a Dell Precision 670 with 4 Gbytes of RAM
- f95 compiler with the -O3 option and ATLAS BLAS and LAPACK
- Comparisons with flagship HSL solver MA57 (Duff)
  - Multifrontal solver (replaced earlier package MA27)
  - Primarily designed for indefinite problems (we use the option to switch off numerical pivoting)

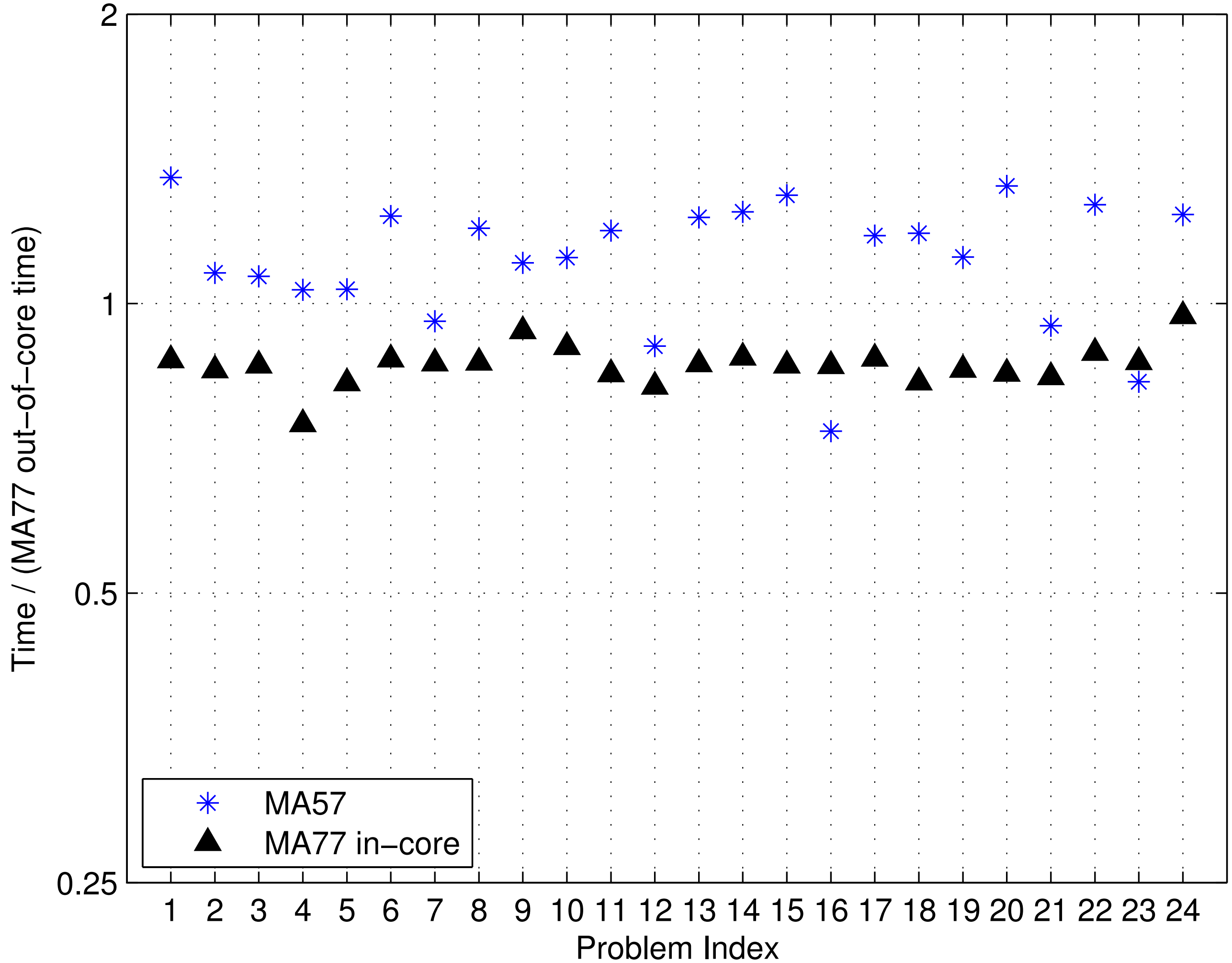
HSL_MA77 factor time compared with MA57



HSL_MA77 solve time compared with MA57



HSL_MA77 total time compared with MA57



Times (in seconds) for large problems

Phase	inline_1 ($n = 503,712$)	bones10 ($n = 914,898$)	nd24k ($n = 72,000$)	bone010 ($n = 986,703$)
Input	4.87	6.25	2.86	8.00
Ordering	14.2	22.8	16.4	34.7
MA77_analyse	4.20	6.70	22.1	26.7
MA77_factor	90.6	174.6	1284	1491
MA77_solve(1)	5.30	36.0	10.4	311
MA77_solve(8)	10.6	41.3	20.7	331
MA77_solve(64)	60.5	141.0	90.2	499

MA57 not able to solve these on our test computer (insufficient memory).

Unsymmetric element problems

- Recently developed out-of-core multifrontal code for unsymmetric element problems. Code is called HSL_MA78
- Based on the design of HSL_MA77
- Again uses HSL_OF01 to handle out-of-core
- Separate package HSL_MA74 written to compute the partial factorization of the dense unsymmetric frontal matrices
  - Implements a block factorization ... employs level 3 BLAS
  - Incorporates threshold pivoting (options for partial, diagonal or rook pivoting)
  - Also option for static pivoting (prevents delayed pivots but may produce inaccurate factorization)
- HSL_MA78 solves $AX = B$ or $A^T X = B$

Comparison with frontal solver

HSL_MA42_ELEMENT is an unsymmetric out-of-core (uni-)frontal code

	n	Time (secs)		Factors ($*10^6$)	
		MA42_ELEMENT	MA78	MA42_ELEMENT	MA78
crplat2	18010	1.85	1.84	4.35	2.94
ship_001	34920	10.5	13.4	15.5	15.6
m_t1	97578	552	101	135.5	56.2
shipsec8	114919	950	101	196.0	55.6
troll	213453	3042	74	672.0	63.7

These results illustrate the benefits of the multifrontal algorithm.

Appeal: We need large test problems in element form from real applications.

Concluding remarks

- Writing these solvers has been (and still is) a major project
- Positive definite code and unsymmetric element code performing well
- Out-of-core working adds an overhead but appears not to be prohibitive (exception is solve phase)
- Indefinite code written, currently testing the indefinite kernel
- Versions for complex arithmetic will be developed

Concluding remarks

- Writing these solvers has been (and still is) a major project
- Positive definite code and unsymmetric element code performing well
- Out-of-core working adds an overhead but appears not to be prohibitive (exception is solve phase)
- Indefinite code written, currently testing the indefinite kernel
- Versions for complex arithmetic will be developed

Out-of-core solvers and virtual memory management package will be in HSL 2007

For details of HSL see `www.cse.scitech.ac.uk/nag/hsl`