Too large to handle directly?

Jennifer Scott

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 $\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^\mathbf{T}$) are popular because they are robust
- But their storage requirements generally grow rapidly with problem size
- One possible solution: use an out-of-core direct solver

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 $\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^\mathbf{T}$) are popular because they are robust
- But their storage requirements generally grow rapidly with problem size
- One possible solution: 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.

Brief history of HSL out-of-core solvers

- MA32 frontal solver for element problems, written by Iain Duff in 1980.
  - Optionally used direct access files to hold the matrix factors.
  - Extended 1983 to assembled unsymmetric systems.
  - Superseded in 1992 by MA42 (Duff and Scott).
    Major change: use of level 3 BLAS.
  - HSL_MA42 (1995) was first HSL package written in Fortran 90.
  - Version for symmetric positive definite systems MA62 in 1996.
  - MPI versions HSL_MP42 and HSL_MP62 (1999).

Brief history of HSL out-of-core solvers

- MA32 frontal solver for element problems, written by Iain Duff in 1980.
  - Optionally used direct access files to hold the matrix factors.
  - Extended 1983 to assembled unsymmetric systems.
  - Superseded in 1992 by MA42 (Duff and Scott).
    Major change: use of level 3 BLAS.
  - HSL_MA42 (1995) was first HSL package written in Fortran 90.
  - Version for symmetric positive definite systems MA62 in 1996.
  - MPI versions HSL_MP42 and HSL_MP62 (1999).

- 25 years ago John Reid developed Cholesky out-of-core multifrontal code TREESOLV for element applications. Good results but never in HSL.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Brief history of HSL out-of-core solvers

- MA32 frontal solver for element problems, written by Iain Duff in 1980.
  - Optionally used direct access files to hold the matrix factors.
  - Extended 1983 to assembled unsymmetric systems.
  - Superseded in 1992 by MA42 (Duff and Scott).
    Major change: use of level 3 BLAS.
  - HSL_MA42 (1995) was first HSL package written in Fortran 90.
  - Version for symmetric positive definite systems MA62 in 1996.
  - MPI versions HSL_MP42 and HSL_MP62 (1999).
- 25 years ago John Reid developed Cholesky out-of-core multifrontal code TREESOLV for element applications. Good results but never in HSL.
- John also wrote out-of-core symmetric positive definite banded solver HSL_MA55 (1999).

Brief history of HSL out-of-core solvers

- MA32 frontal solver for element problems, written by Iain Duff in 1980.
  - Optionally used direct access files to hold the matrix factors.
  - Extended 1983 to assembled unsymmetric systems.
  - Superseded in 1992 by MA42 (Duff and Scott).
    Major change: use of level 3 BLAS.
  - HSL_MA42 (1995) was first HSL package written in Fortran 90.
  - Version for symmetric positive definite systems MA62 in 1996.
  - MPI versions HSL_MP42 and HSL_MP62 (1999).
- 25 years ago John Reid developed Cholesky out-of-core multifrontal code TREESOLV for element applications. Good results but never in HSL.
- John also wrote out-of-core symmetric positive definite banded solver HSL_MA55 (1999).

TREESOLV is really the inspiration for our recent codes.

Other out-of-core solvers

- `BCSEXT-LIB` (Boeing)
- `Oblio` (Dobrian and Pothen)
- `TAUCS` (Toledo and students)
- `MUMPS` parallel solver: now offers out-of-core version
- Also work by Rothberg and Schreiber

HSL_MA77

HSL_MA77 is our new out-of-core solver

- HSL_MA77 is designed to solve **LARGE** sparse symmetric systems, both positive definite and indefinite
- HSL_MA77 implements a **multifrontal algorithm**
- 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 (recall: HSL is a library of Fortran packages)

Key features of HSL_MA77

- Written in Fortran 95 (recall: HSL is a library of Fortran packages)
- Matrix $\mathbf{A}$ may be either in assembled form or a sum of element matrices

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Key features of HSL_MA77

- Written in Fortran 95 (recall: HSL is a library of Fortran packages)
- 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 (recall: HSL is a library of Fortran packages)
- 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

Key features of HSL_MA77

- Written in Fortran 95 (recall: HSL is a library of Fortran packages)
- 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)
  - Compute residual and obtain information on factors (optional)
  - Optional restart (save data for later factorization and/or solves)
  - Optional scaling (out-of-core)

Key features of HSL_MA77

- Written in Fortran 95 (recall: HSL is a library of Fortran packages)
- 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
- Additional flexibility through user-controlled parameters (default settings minimize decisions user must make)

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices
- We have developed separate packages to perform these factorizations (and partial solves)
  - `HSL_MA54` for positive definite problems
  - `HSL_MA64` for indefinite problems (uses threshold partial pivoting with 1x1 and 2x2 pivots)

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices
- We have developed separate packages to perform these factorizations (and partial solves)
  - `HSL_MA54` for positive definite problems
  - `HSL_MA64` for indefinite problems (uses threshold partial pivoting with 1x1 and 2x2 pivots)
- Kernels use blocking and exploit Level 3 BLAS. OpenMP option now available

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices
- We have developed separate packages to perform these factorizations (and partial solves)
  - `HSL_MA54` for positive definite problems
  - `HSL_MA64` for indefinite problems (uses threshold partial pivoting with 1x1 and 2x2 pivots)
- Kernels use blocking and exploit Level 3 BLAS. OpenMP option now available
- Modular design helps with readability, testing, maintenance etc

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices
- We have developed separate packages to perform these factorizations (and partial solves)
  - HSL_MA54 for positive definite problems
  - HSL_MA64 for indefinite problems (uses threshold partial pivoting with 1x1 and 2x2 pivots)
- Kernels use blocking and exploit Level 3 BLAS. OpenMP option now available
- Modular design helps with readability, testing, maintenance etc
- Kernels can also be reused in other solvers

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Dense linear algebra kernels

- At the heart of the multifrontal method is the partial factorization of dense frontal matrices
- We have developed separate packages to perform these factorizations (and partial solves)
  - `HSL_MA54` for positive definite problems
  - `HSL_MA64` for indefinite problems (uses threshold partial pivoting with 1x1 and 2x2 pivots)
- Kernels use blocking and exploit Level 3 BLAS. OpenMP option now available
- Modular design helps with readability, testing, maintenance etc
- Kernels can also be reused in other solvers
- Performance can be tuned for computing environment

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

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

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)
- 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 data from stack

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)
- factorization phase
- 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 - direct access 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
- Note: we do not want to be restricted to only accessing the data in the same order as it was written so sequential access not an option

Input/Output in Fortran

In Fortran 77/90/95 - direct access 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
- Note: we do not want to be restricted to only accessing the data in the same order as it was written so sequential access not an option

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.
Note: John wrote an earlier code `OF01` for use by `TREESOLV`.

Virtual memory management

We have a separate Fortran 95 package `HSL_OF01` that handles all i/o.
Note: John wrote an earlier code `OF01` for use by `TREESOLV`.

- `HSL_OF01` provides read/write facilities for one or more direct access files through a single **in-core buffer** (work array)

Virtual memory management

We have a separate Fortran 95 package `HSL_OF01` that handles all i/o.
Note: John wrote an earlier code `OF01` for use by `TREESOLV`.

- `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.

Virtual memory management

We have a separate Fortran 95 package `HSL_OF01` that handles all i/o.
Note: John wrote an earlier code `OF01` for use by `TREESOLV`.

- `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

Virtual memory management

We have a separate Fortran 95 package `HSL_OF01` that handles all i/o.
Note: John wrote an earlier code `OF01` for use by `TREESOLV`.

- `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 eg
  - All wanted pages that are in buffer are accessed before those that are not
  - When a page is freed, only written to file if it has changed

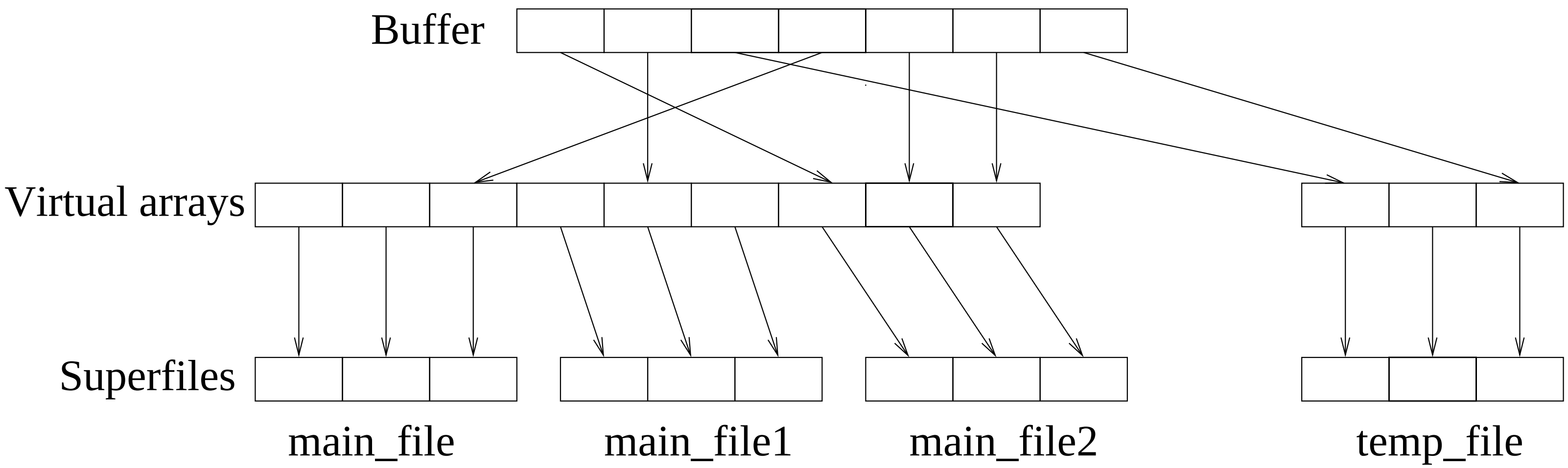
Virtual memory management

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
- Most active pages of the virtual array are held in the buffer
- 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 (or three) files:
  - one holds the real data for the matrix **A** and the matrix factor
  - another is used for the multifrontal stack (work space)
  - in the indefinite case, third file holds separate multifrontal stack for data from delayed pivots
- The user supplies pathnames together with names for the primary files

Use of HSL_OF01 within HSL_MA77

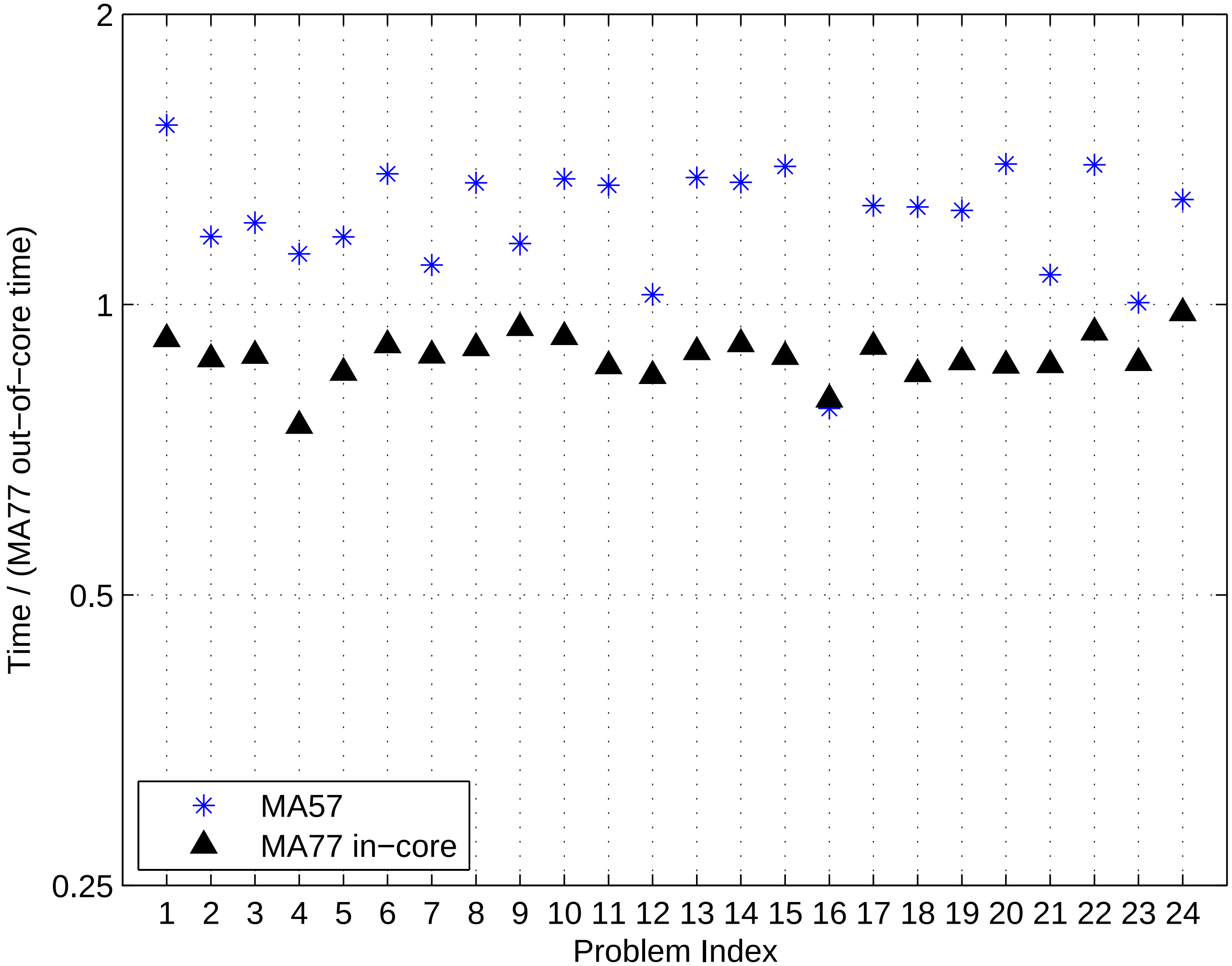
- HSL_MA77 has one **integer** buffer and one **real** buffer
- The integer buffer is associated with a file that holds the integer data for the matrix **A** and the matrix factor
- The real buffer is associated with two (or three) files:
  - one holds the real data for the matrix **A** and the matrix factor
  - another is used for the multifrontal stack (work space)
  - in the indefinite case, third file holds separate multifrontal stack for data from 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 if enough memory available). A combination of files and arrays may be used.

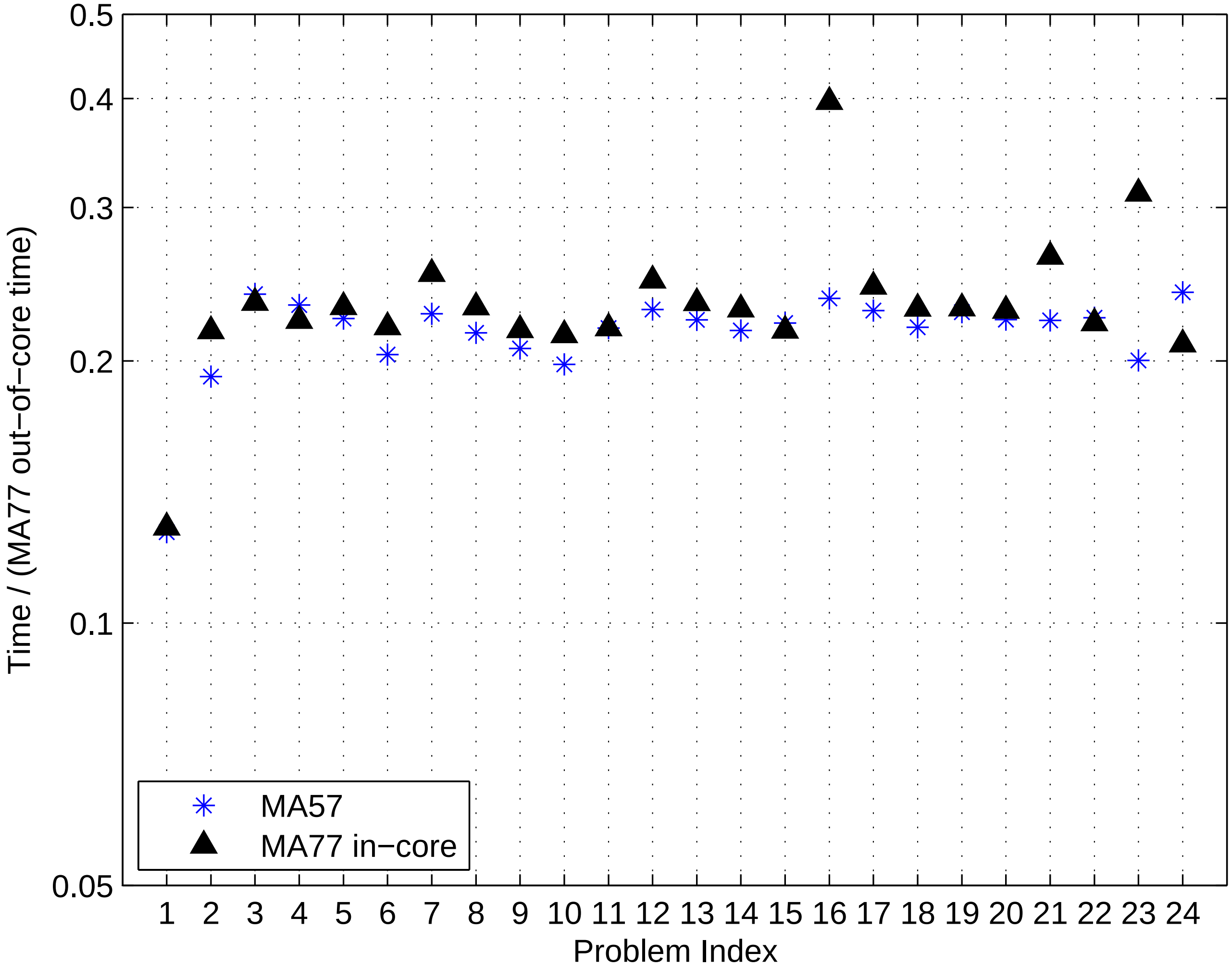
Some numerical experiments

- Large problems from University of Florida Sparse Matrix Collection
- Double precision (64-bit) reals on single processor of a Dell Precision 670 with two 3.6 GHz Intel Xeon processors and 4 Gbytes of RAM
- f95 compiler with the -O3 option and ATLAS BLAS and LAPACK
- Comparisons with HSL solver `MA57` (recall yesterday's talk)

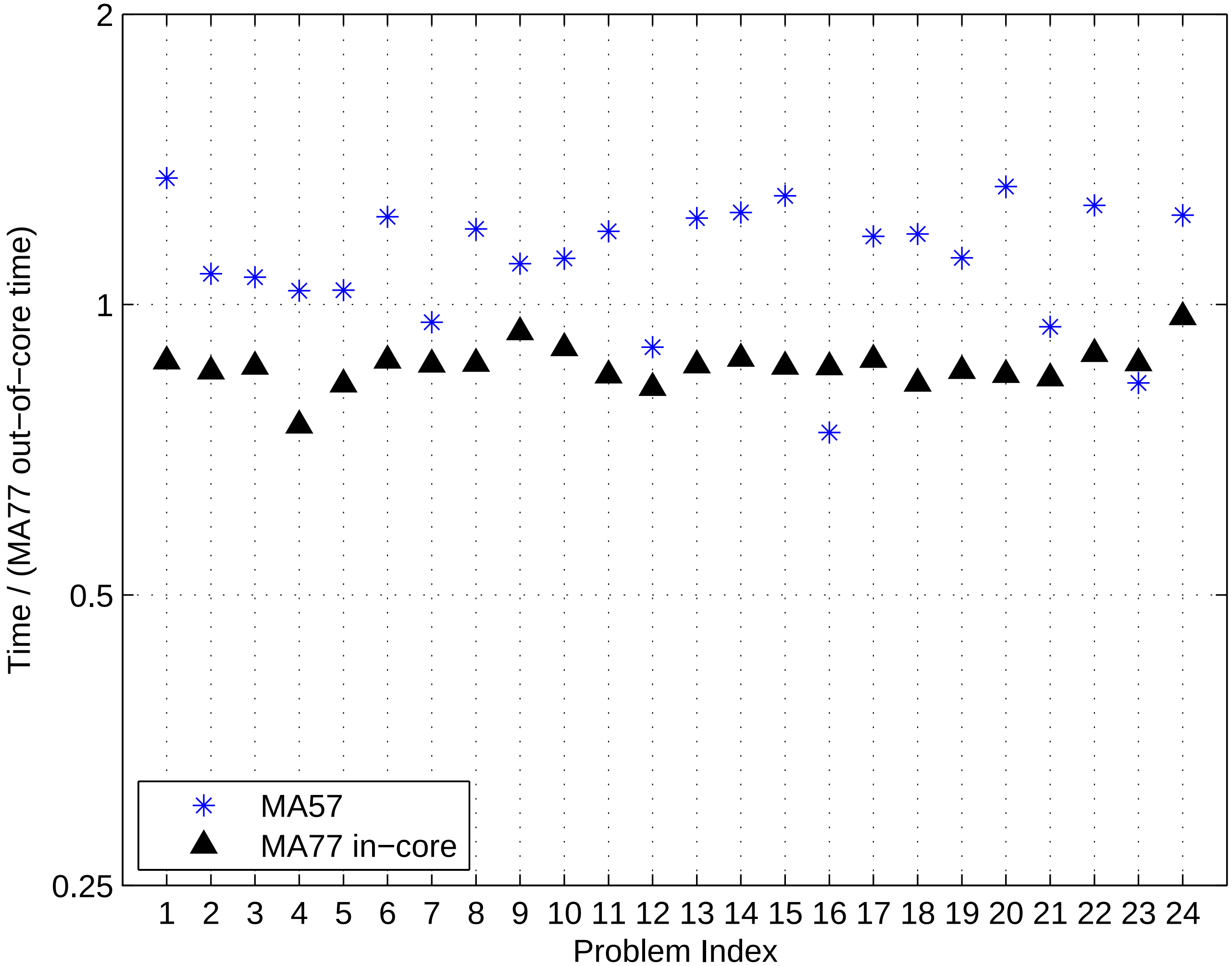
Factor time: positive definite problems



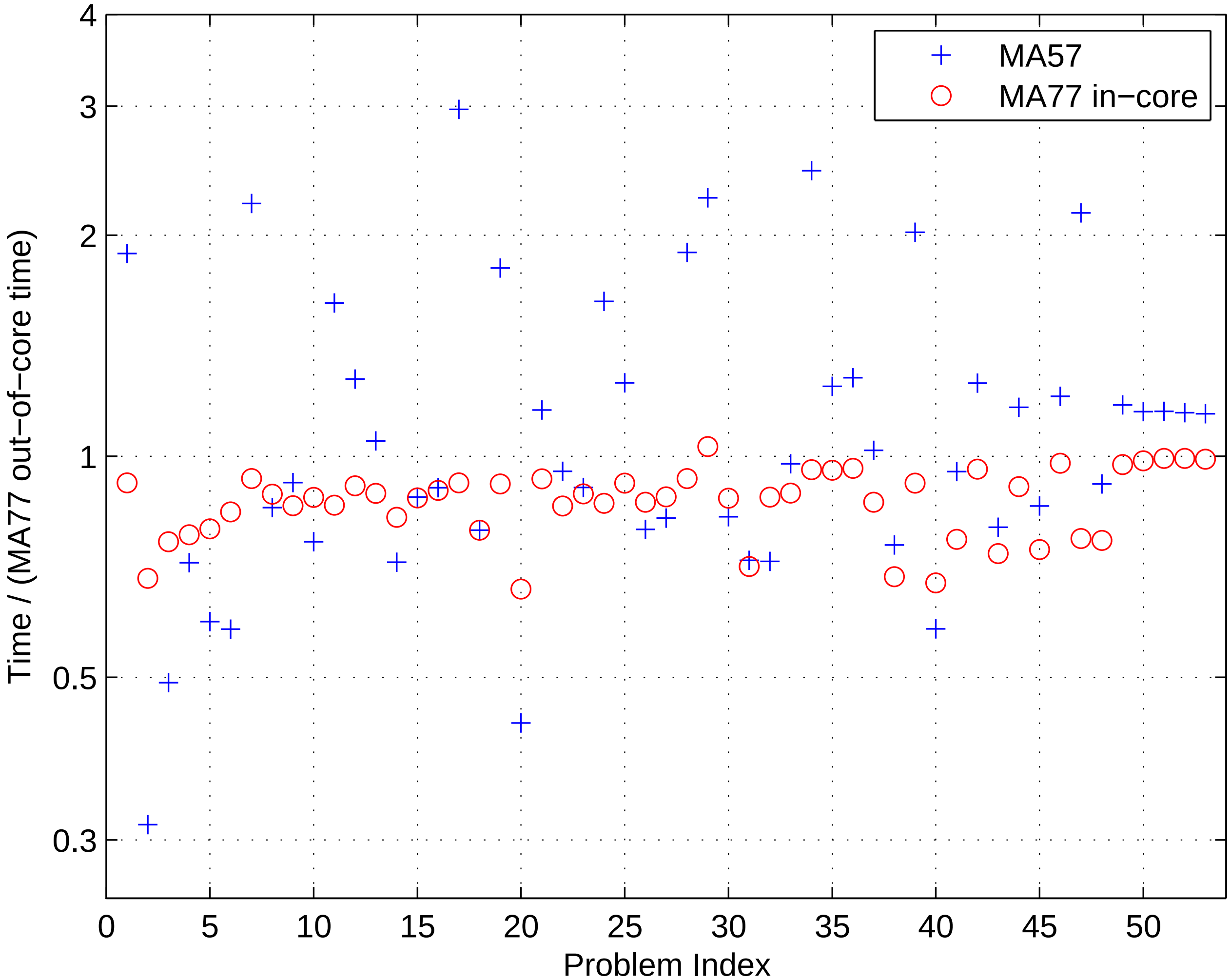
Solve time: positive definite problems



Total time: positive definite problems



Total time: indefinite problems



Times (seconds) for large problems

Phase	inline_1 (503,712)	bones10 (914,898)	nd24k (72,000)	bone010 (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	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	90.2	499

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

Unsymmetric element problems

- Also 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 kernel routine `HSL_MA74` computes the partial factorization of the dense unsymmetric frontal matrices

Comparison with frontal solver

	n	Time (secs)		Factors ($*10^6$)	
		MA42_ELEMENT	MA78	MA42_ELEMENT	MA78
ship_001	34920	10.5	13.4	15.5	15.6
m_t1	97578	552	101	135	56.2
shipsec8	114919	950	101	196	55.6
troll	213453	3102	68	671	63.7
fullb	199187	786	80	356	75.1

These results illustrate the benefits of the multifrontal algorithm.

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

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Pivoting options

HSL_MA78 offers threshold **partial** pivoting and threshold **rook** pivoting.

Threshold partial pivot: candidate must satisfy

$$|f_{ij}| \geq \mathtt{u} * \max |f_{lj}|$$

where $0 \leq \mathtt{u} \leq 1$ is the pivoting *threshold* parameter.

Threshold rook pivot: candidate must also satisfy

$$|f_{ij}| \geq \mathtt{u} * \max |f_{il}|$$

ie threshold test in both **row** and **columns**.

More expensive but more stable (controls condition of L and U).

Does it pay off?

Rook versus partial pivoting

	n	Time (secs)		Factors ($*10^6$)		Residual	
		rook	partial	rook	partial	rook	partial
ship_001	34920	15.0	13.4	15.6	15.6	$5.7 * 10^{-16}$	$3.1 * 10^{-16}$
m_t1	97578	**55.7**	94.9	40.2	56.2	$4.7 * 10^{-16}$	$8.5 * 10^{-14}$
shipsec5	179860	**175**	275	80.4	105	$1.8 * 10^{-15}$	$6.8 * 10^{-13}$
ship_003	121178	146	**118**	70.8	74.0	$7.9 * 10^{-16}$	$1.5 * 10^{-13}$
raju_001	151656	335	**226**	168	147	$1.5 * 10^{-15}$	$5.8 * 10^{-15}$

Conclude: rook pivoting can be beneficial.

Out-of-core scaling

$$\tilde{A} = D_R^{-1} A D_C^{-1}$$

where D_R, D_C diagonal matrices, is an **equilibration** of A if norms of its rows and columns have approx. same magnitude.

One possibility (Ruiz)

$$D_R = \mathrm{diag}\left(\sqrt{\max_j |A_{ij}|}\right) \text{ and } D_C = \mathrm{diag}\left(\sqrt{\max_i |A_{ij}|}\right).$$

May be applied iteratively.

Can we implement this without explicitly assembling A?

Out-of-core scaling

- Recall: each stage of multifrontal method involves a frontal matrix

$$F = \begin{pmatrix} F_1 & F_2 \\ F_3 & F_4 \end{pmatrix}$$

- p rows of F_1 and F_2 are fully summed.
- p columns of F_1 and F_3 are fully summed.
- Search first p rows and columns of F and accumulate the largest entries

Out-of-core scaling

- Suppose row i of F corresponds to global row k of A
- If $i \leq p$

$$(D_R)_k \leftarrow \max\left((D_R)_k,\ \max_{j \leq m} |f_{ij}|\right)$$

Otherwise

$$(D_R)_k \leftarrow \max\left((D_R)_k,\ \max_{j \leq p} |f_{ij}|\right)$$

- Similarly for $(D_C)_k$.
- Update D_R and D_C and then discard F_1, F_2, F_3 and stack F_4.
- Continue to next node of tree
- Avoids assembly A in main memory **but** does require significant I/O

Effects of equilibration

	Rook		Partial	
	No scaling	Scaling	No scaling	Scaling
x104	34.0	24.9	37.8	**23.0**
m_t1	55.7	**45.0**	94.9	63.0
shipsec1	110	49.3	174	**44.4**
thread	37.8	55.0	**35.4**	64.6

Notes:

- Scaling adds overhead and may not give benefit.
- **But** scaling can cut total cost and can be particularly beneficial for partial pivoting.
- Scaled residuals typically an order of magnitude large for partial pivoting.

Concluding remarks

- New HSL direct solvers are performing well on large problems
- Able to solve larger problems than previously on desktop machines
- Out-of-core working adds an overhead but our memory management system attempts to minimise this (note: single rhs solve expensive)
- Scaling out of core is a new development
- Rook pivoting looks to be a useful option
- Recently we have looked at developing parallel version but out-of-core working adds complications
- Reports giving full details are available via our webpages

Concluding remarks

- New HSL direct solvers are performing well on large problems
- Able to solve larger problems than previously on desktop machines
- Out-of-core working adds an overhead but our memory management system attempts to minimise this (note: single rhs solve expensive)
- Scaling out of core is a new development
- Rook pivoting looks to be a useful option
- Recently we have looked at developing parallel version but out-of-core working adds complications
- Reports giving full details are available via our webpages

Thank you and thank you John!