An introduction to the world of sparse direct solvers

Jennifer Scott, STFC Rutherford Appleton Laboratory

AT&T December 2011

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Numerical Analysis Group at RAL

Based at the Rutherford Appleton Laboratory in rural Oxfordshire (about 15 miles south of Oxford).

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Numerical Analysis Group at RAL

Small research group: Jennifer Scott (leader), Iain Duff, Nick Gould, Mario Arioli, Sue Thorne, Jonathan Hogg, John Reid (Honorary Scientist).

Science & Technology Facilities Council
Rutherford Appleton Laboratory

HSL

A key activity of the NA Group is the development, maintenance of the mathematical software library HSL.

HSL began as Harwell Subroutine Library in 1963.

Collection of portable, fully documented and tested Fortran packages (some Matlab and C interfaces).

Each package performs a basic numerical task (eg solve linear system, find eigenvalues) and is designed to be incorporated into programs.

Since 1970s, a key strength of HSL and one for which is internationally known is sparse matrix computations.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Sparse matrices: brief introduction

Problem: we wish to solve

$$Ax = b$$

where A is

LARGE

s p a r s e

What is sparse? A is sparse if

- many entries are zero
- it is worthwhile to exploit these zeros.

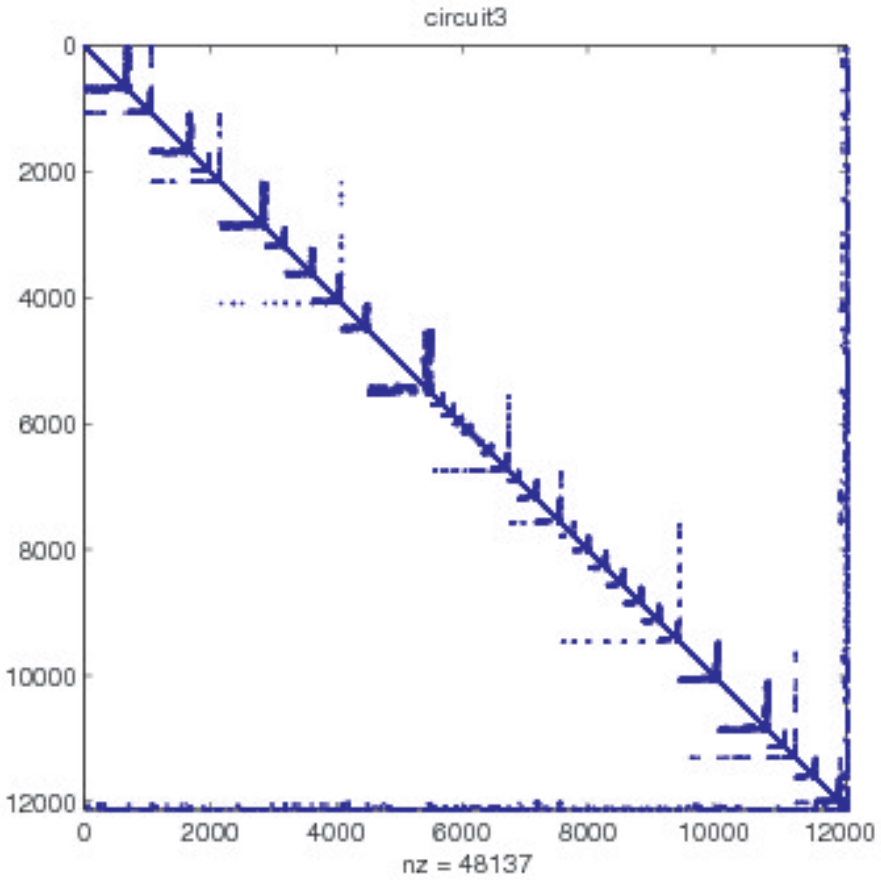
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Sparse matrices: applications

Many application areas in science, engineering, and finance eg.

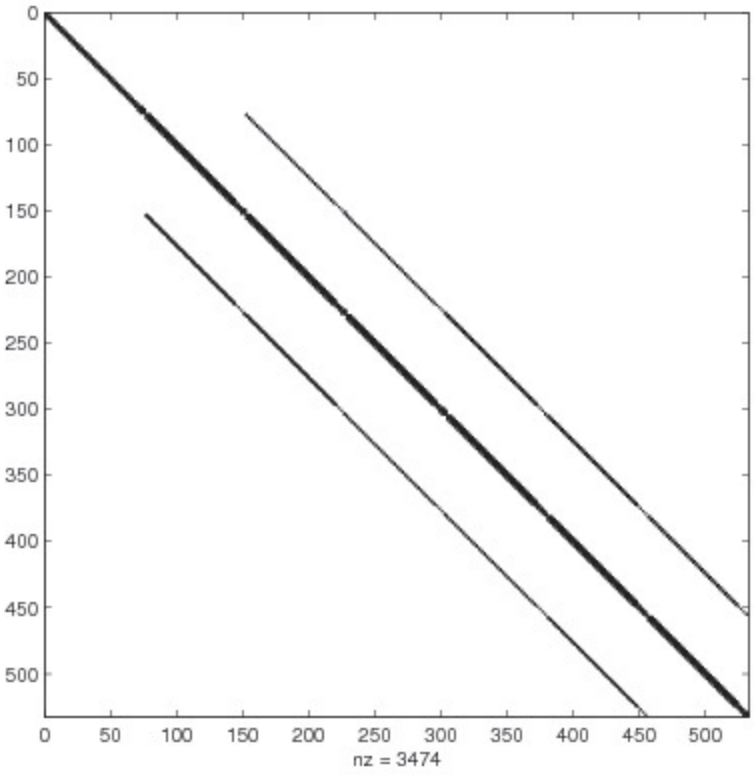
- computational fluid dynamics
- chemical engineering
- circuit simulation
- economic modelling
- fluid flow
- oceanography
- linear programming
- structural engineering ...

Circuit simulation



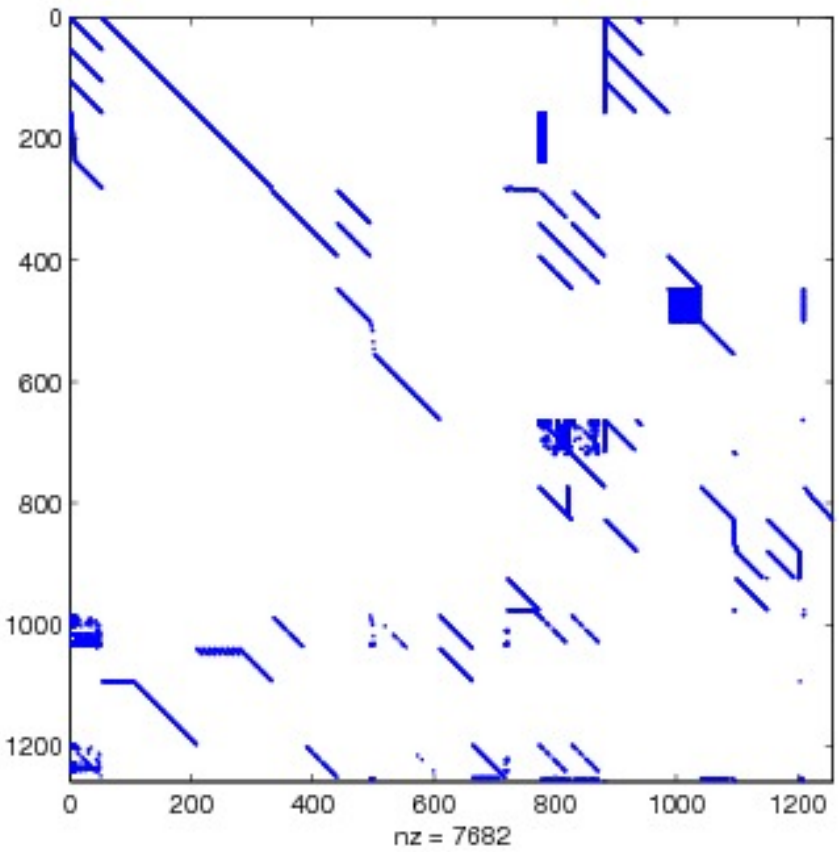
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Reservoir modelling



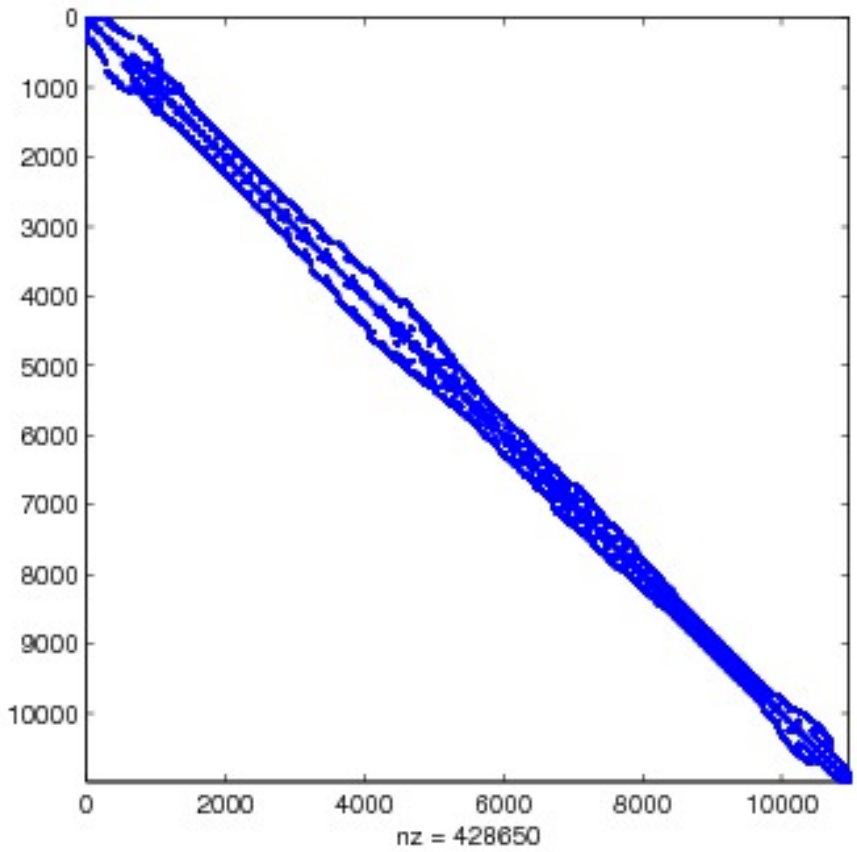
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Economic modelling



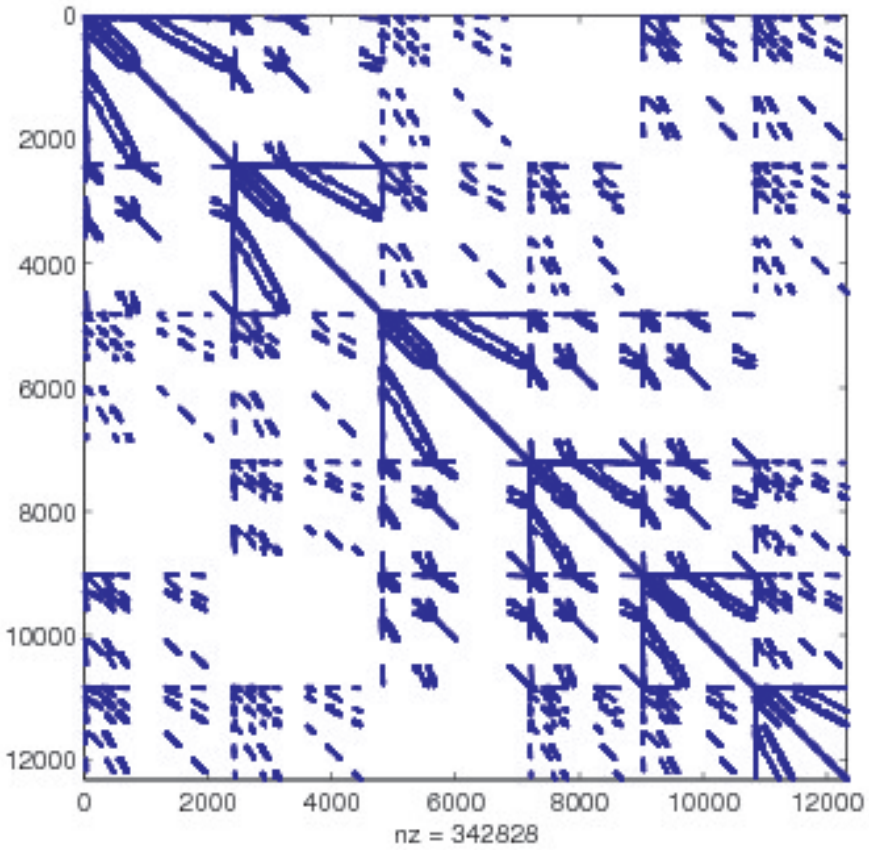
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Structural engineering



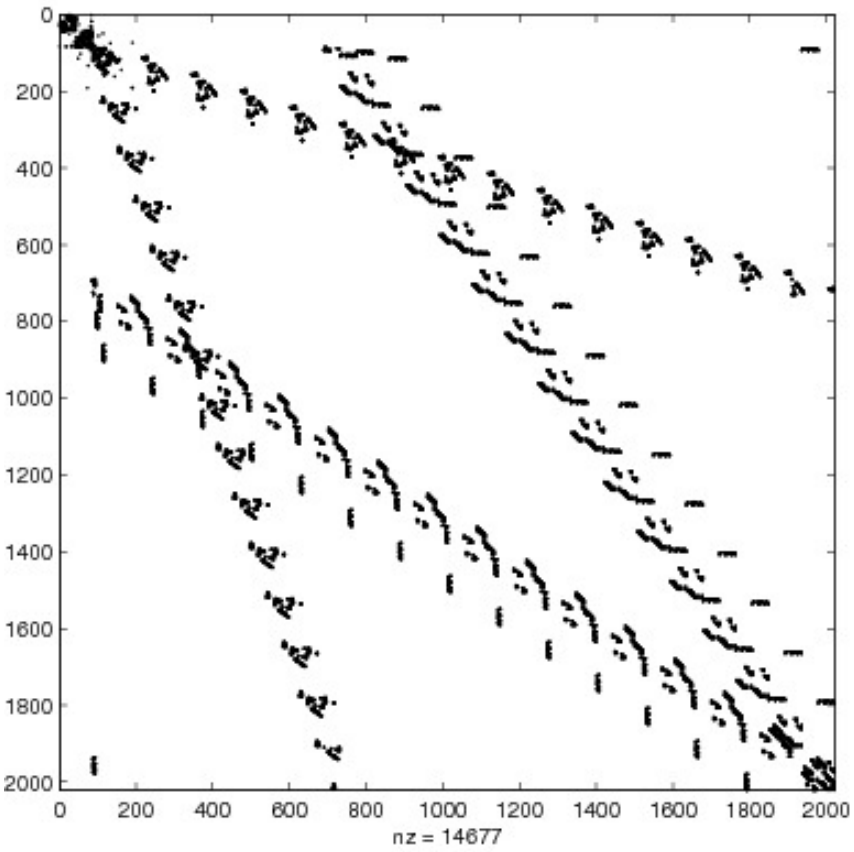
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Acoustics



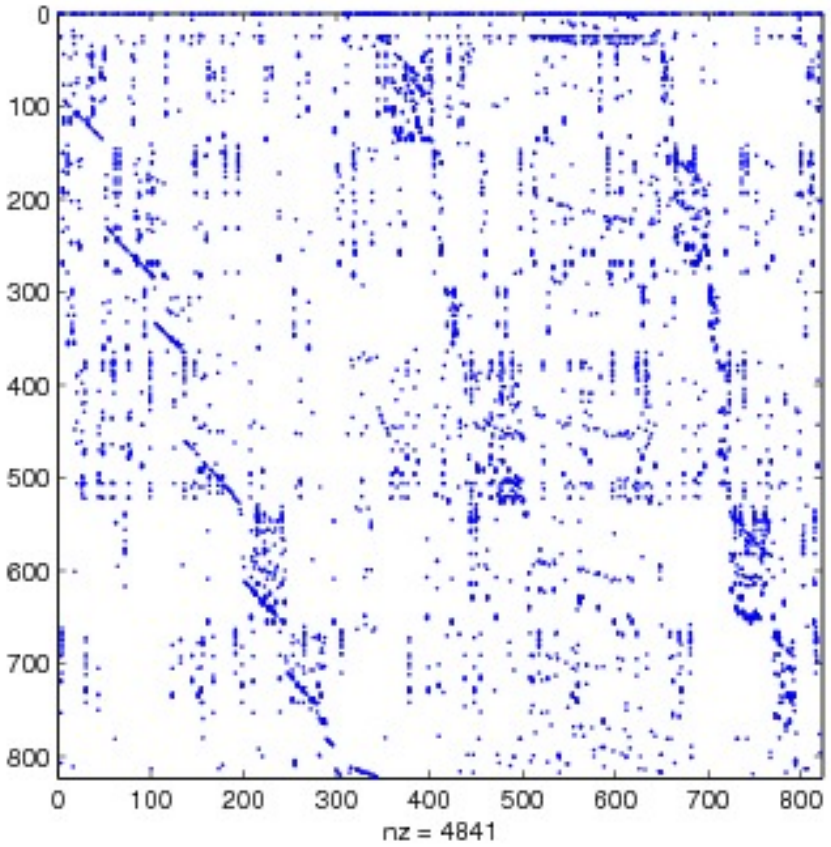
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Chemical engineering



Science & Technology Facilities Council
Rutherford Appleton Laboratory

Linear programming



Solving sparse linear systems

Two main classes of methods:

- **Direct methods** are usually variants of Gaussian elimination and involve explicit factorization eg $PAQ = LU$
  - L, U lower and upper triangular matrices
  - P, Q are permutation matrices
  - Solution process completed by (easy) triangular solves $Ly = Pb$ and $Uz = y$ then $x = Qz$
- **Iterative methods** eg conjugate gradients, GMRES, BiCGSTAB, MINRES ...

Direct methods

Advantages:

- High accuracy.
- Robust. Can be used as black box solvers .
- Solving for multiple right-hand sides almost as cheap as one right hand side.

Disdvantages:

- Memory required grows more rapidly than problem size.
- Difficult to code efficiently. (Massive) parallelism very hard.

Iterative methods

Advantages

- Need only a small number of arrays of length n.
- Easy to code.
- Speed depends on matrix-vector products ... parallelise.
- Can choose accuracy.

Disdvantages

- Lack of robustness.
- Require preconditioner but how to choose? Highly problem dependent. Difficult in parallel.
- May want to solve for many right hand sides.

Today's talk

Today we are interested in **direct solvers**.

But note that direct methods are often used to compute preconditioners for iterative solvers eg ILU.

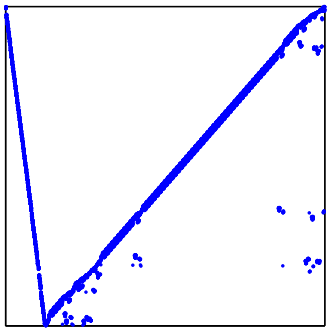
Phases of a direct solvers

Sparse direct solvers have a number of distinct phases, typically

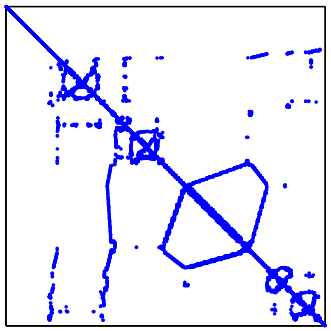
- **ORDER:** preorder the matrix to exploit structure
- **ANALYSE:** analyse matrix structure to produce data structures for factorization
- **FACTORIZE:** perform numerical factorization
- **SOLVE:** use factors to solve one or more systems

Effect of ordering

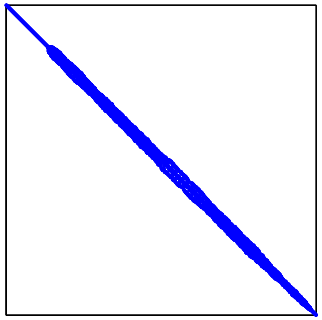
extr1



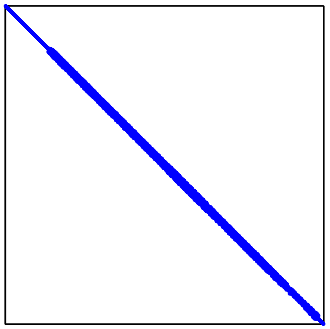
Block form



$A+A^T$



Unsymmetric



Outline of rest of talk

We will focus on the factorization phase (generally most expensive part of a direct solver).

How to efficiently solve $A\mathbf{x} = \mathbf{b}$ on multicore machines

- Dense positive-definite systems
- Sparse positive-definite systems
- Sparse indefinite systems
- Concluding remarks

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Solving systems in parallel

We want to solve

- Medium and large problems (more than 10^{10} flops)
- On desktop machines (multicore)
- Shared memory, complex cache-based architectures

Solving systems in parallel

We want to solve

- Medium and large problems (more than 10^{10} flops)
- On desktop machines (multicore)
- Shared memory, complex cache-based architectures

I have an 8-core machine...

...I want to go (nearly) 8 times faster

Dense systems

Let's start with dense systems.

We want to solve

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

where the $n \times n$ matrix A is

- symmetric and dense
- positive definite (indefinite problems require pivoting)
- not small (order at least a few hundred)

Pen and paper approach

Factorize $A = LL^T$ then solve $A\mathbf{x} = \mathbf{b}$ as

$$\begin{aligned} L\mathbf{y} &= \mathbf{b} \\ L^T\mathbf{x} &= \mathbf{y} \end{aligned}$$

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Pen and paper approach

Factorize $A = LL^T$ then solve $A\mathbf{x} = \mathbf{b}$ as

$$\begin{aligned} L\mathbf{y} &= \mathbf{b} \\ L^T\mathbf{x} &= \mathbf{y} \end{aligned}$$

Algorithm:

- For each column k:
  - $L_{kk} = \sqrt{A_{kk}}$ (Calculate diagonal element)
  - For rows $i > k$: $L_{ik} = A_{ik}L_{kk}^{-1}$ (Divide column by diagonal)
  - Update trailing submatrix $A_{(k+1:n)(k+1:n)} \leftarrow A_{(k+1:n)(k+1:n)} - L_{(k+1:n)k}L_{(k+1:n)k}^T$

Serial approach

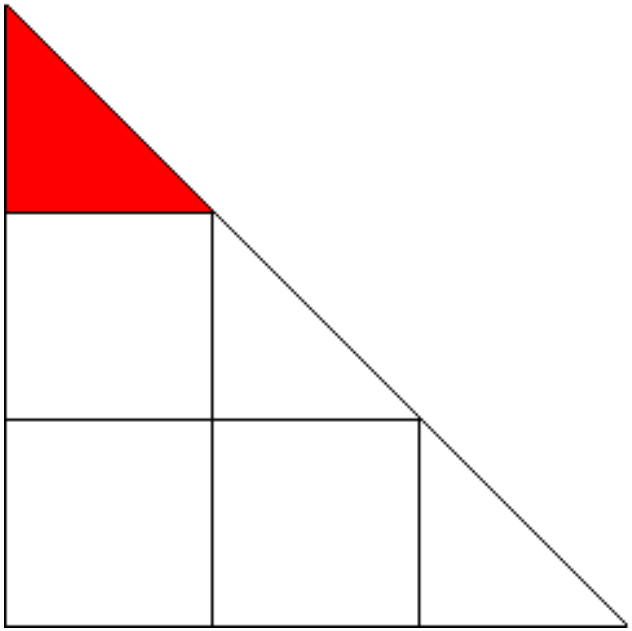
Aim to exploit caches

Work with blocks

- Same algorithm, but submatrices not elements
  - Factor: $A_k = L_{kk} L_{kk}^T$
  - Solve: $L_{ik} = A_{ik} L_{kk}^{-1}$
  - Update: $A_{ij} \leftarrow A_{ij} - L_{ik} L_{kj}^T$
- 10$\times$ faster than a naive implementation
- Built using Level 3 Basic Linear Algebra Subroutines (BLAS) eg `gemm` for the update operations

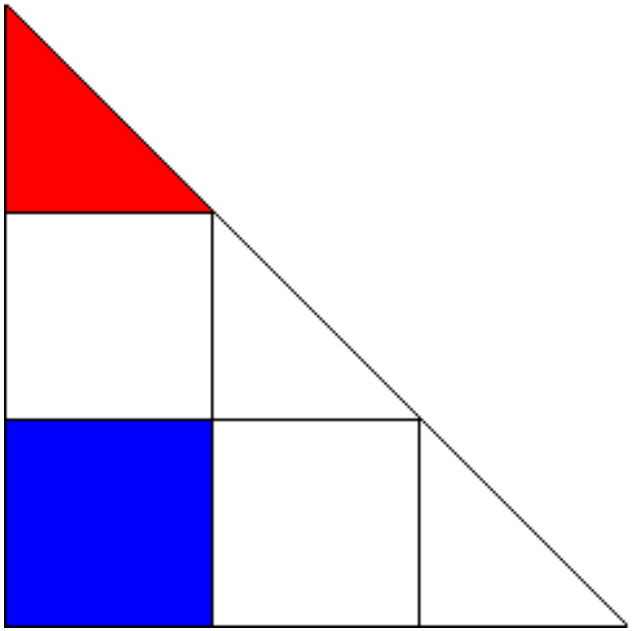
Cholesky by blocks

Factorize diagonal



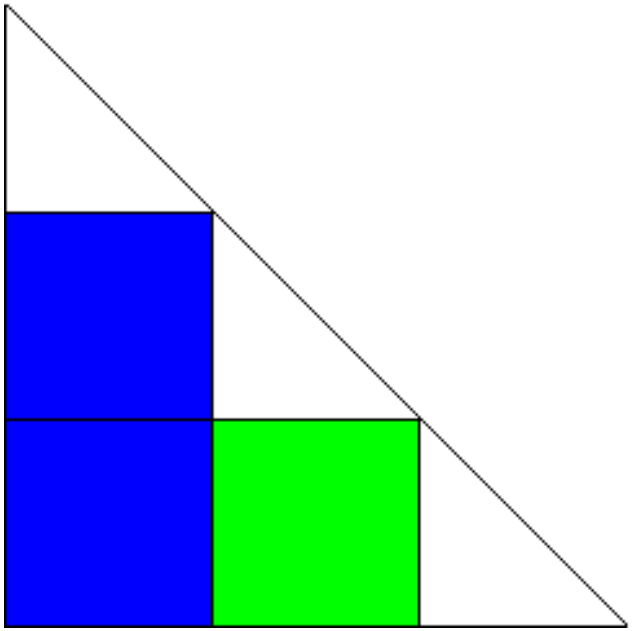
Cholesky by blocks

Solve column block



Cholesky by blocks

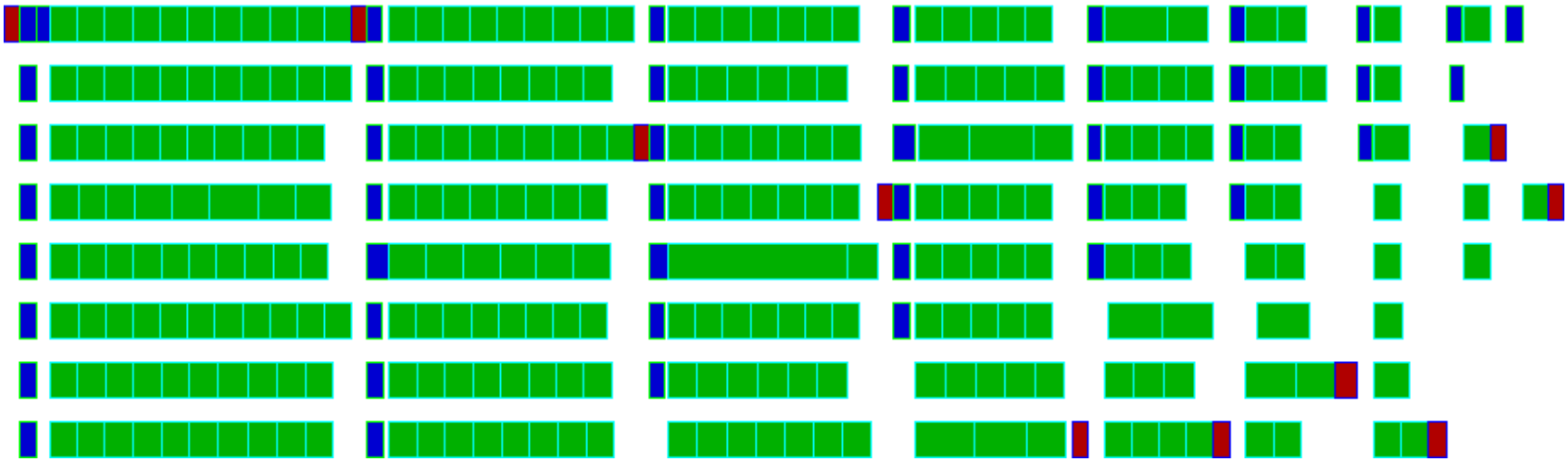
Update block



Simple approach to parallelism

Parallel right-looking algorithm

red is factor; blue is solve; green is update



DAGs

What do we really need to synchronise?

DAGs

What do we really need to synchronise?

Represent each block operation (Factor, Solve, Update) as a task.

Tasks have dependencies.

DAGs

What do we really need to synchronise?

Represent each block operation (Factor, Solve, Update) as a task.

Tasks have dependencies.

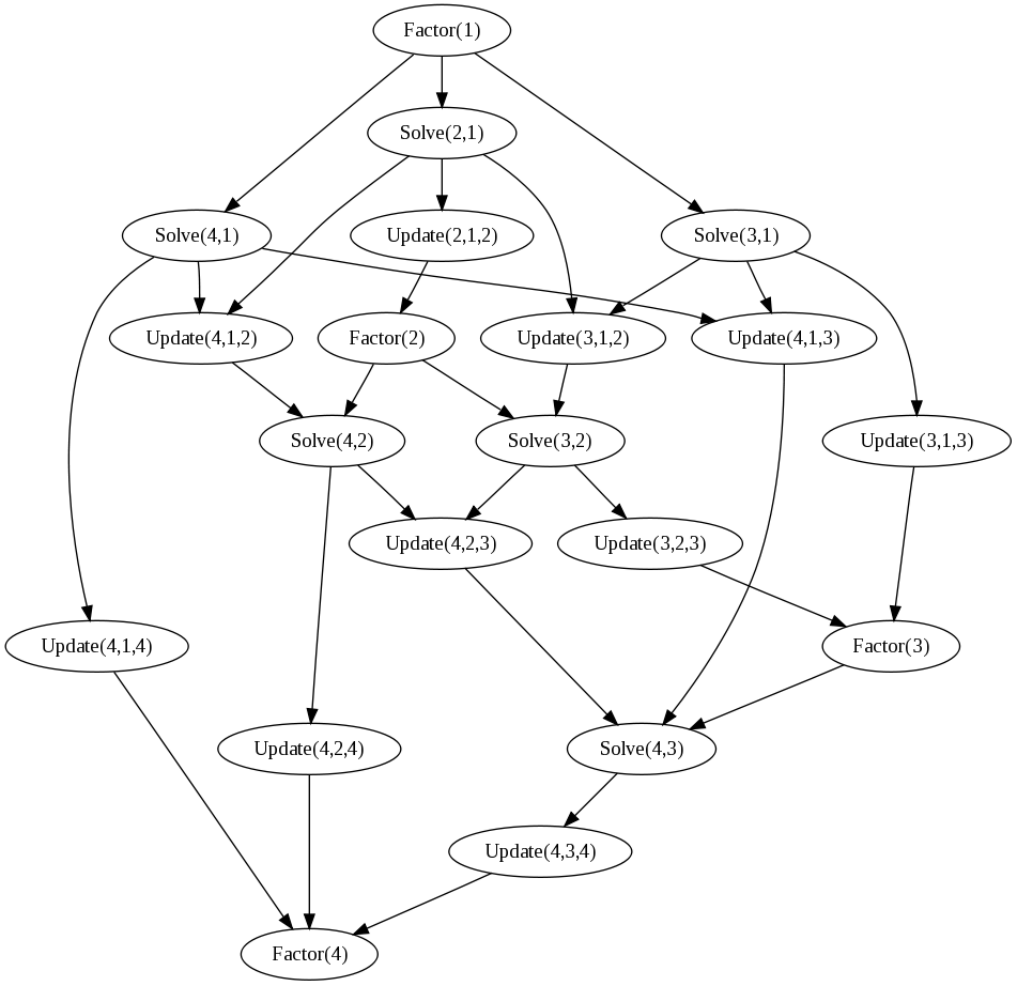
Represent this as a directed graph

- Tasks are vertices
- Dependencies are directed edges

It is acyclic — hence have a Directed Acyclic Graph (DAG).

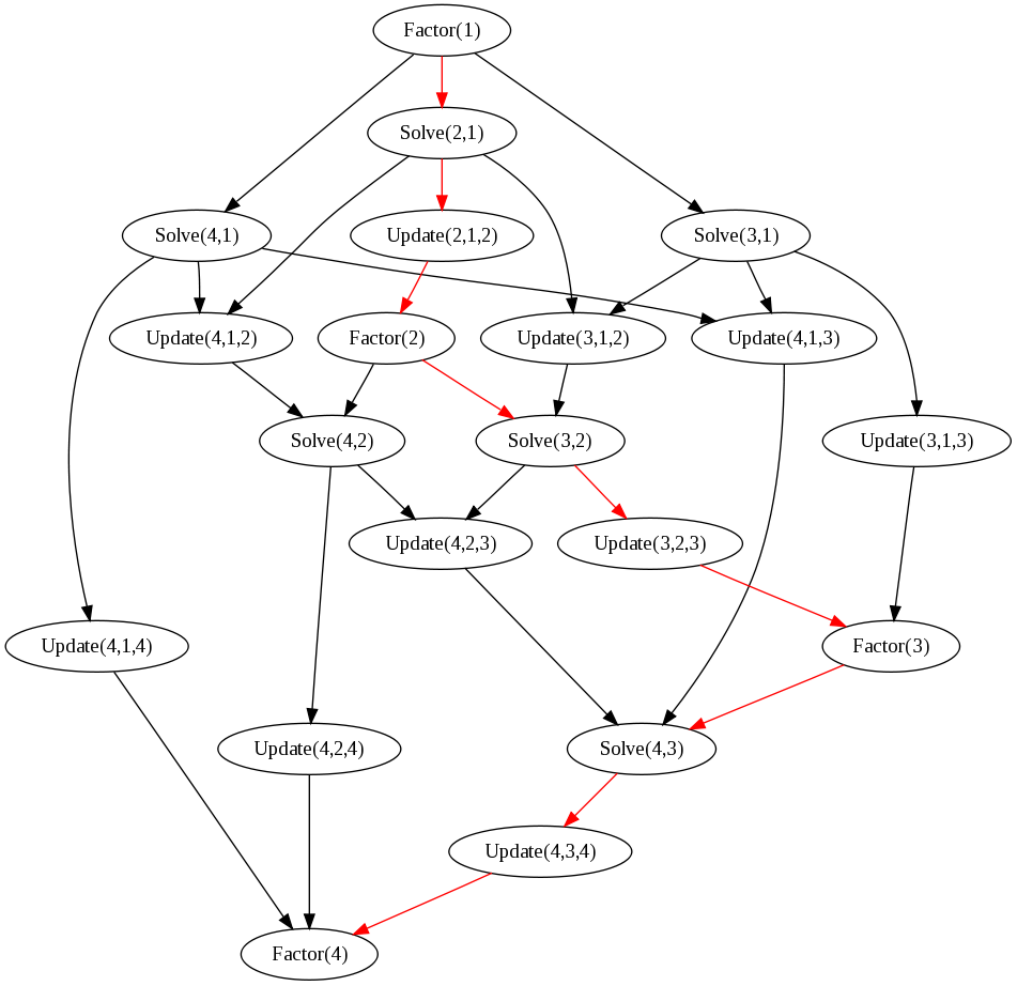
Used by (eg) Buttari, Dongarra, Kurzak, Langou, Luszczek, Tomov ('06)

Task DAG



Science & Technology Facilities Council
Rutherford Appleton Laboratory

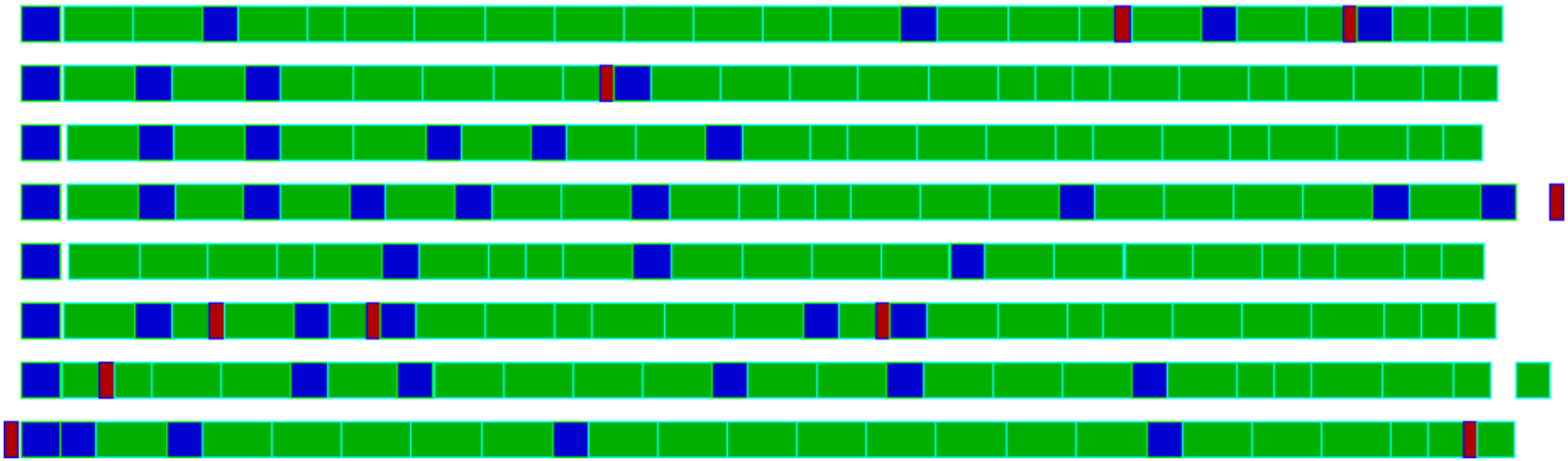
Task DAG



Science & Technology Facilities Council
Rutherford Appleton Laboratory

Profile for DAG approach

Using DAG approach, white space (idle time) mostly disappears.



Performance in Gflop/s for dense case

8 core machine, peak performance of gemm is 72.8

threads	1	2	4	8	speedup
$n = 500$	5.6	8.6	13.4	17.7	3.2
2500	7.6	14.5	26.9	43.5	5.7
10000	8.6	17.1	33.6	61.9	7.2
20000	8.8	17.7	35.1	65.5	7.4

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Sparse case?

So far, so dense. What about sparse factorizations?

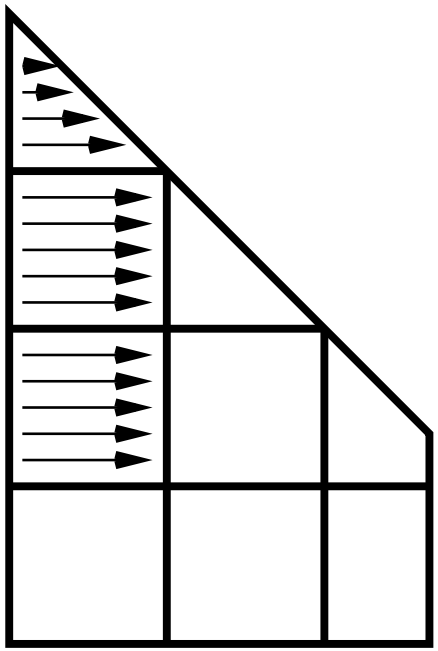
Sparse matrices

- Sparse matrix is mostly zero — only track non-zeros.
- Factor L is denser than A.
- Extra entries are known as fill-in.
- Reduce fill-in by preordering A.

Aim: Organise computations to use dense kernels on submatrices.

Nodal matrices

Assuming null rows have been removed, a block column of L is stored as a dense submatrix



Sparse L is made up of many of these dense block columns

Sparse DAG

Basic idea: Extend DAG-based approach to the sparse case by adding new type of task to perform sparse update operations.

Sparse DAG

Basic idea: Extend DAG-based approach to the sparse case by adding new type of task to perform sparse update operations.

Tasks in sparse DAG:
factor(`diag`): performs dense Cholesky factorization of the block `diag` on diagonal

$$L_{diag} = L_t L_t^T$$

Sparse DAG

Basic idea: Extend DAG-based approach to the sparse case by adding new type of task to perform sparse update operations.

Tasks in sparse DAG:
factor(`diag`): performs dense Cholesky factorization of the block `diag` on diagonal

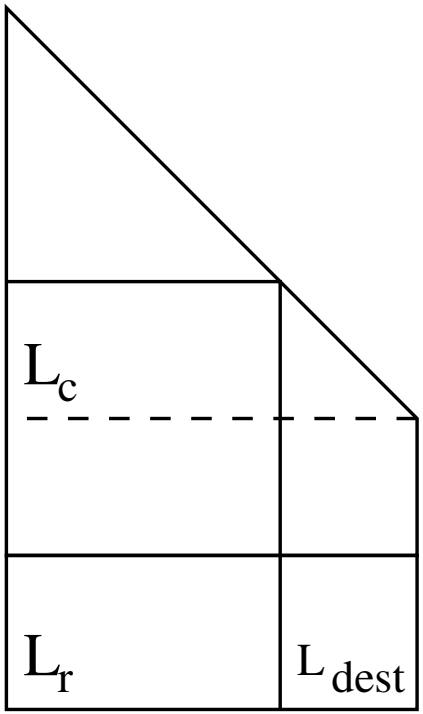
$$L_{diag} = L_t L_t^T$$

solve(`dest`, `diag`): performs triangular solve of off-diagonal block `dest` by Cholesky factor L_t of block `diag` on its diagonal

$$L_{dest} \Leftarrow L_{dest} L_t^{-T}$$

Tasks in sparse DAG

update_internal: performs update within nodal matrix,



$$L_{dest} \Leftarrow L_{dest} - L_r L_c^T$$

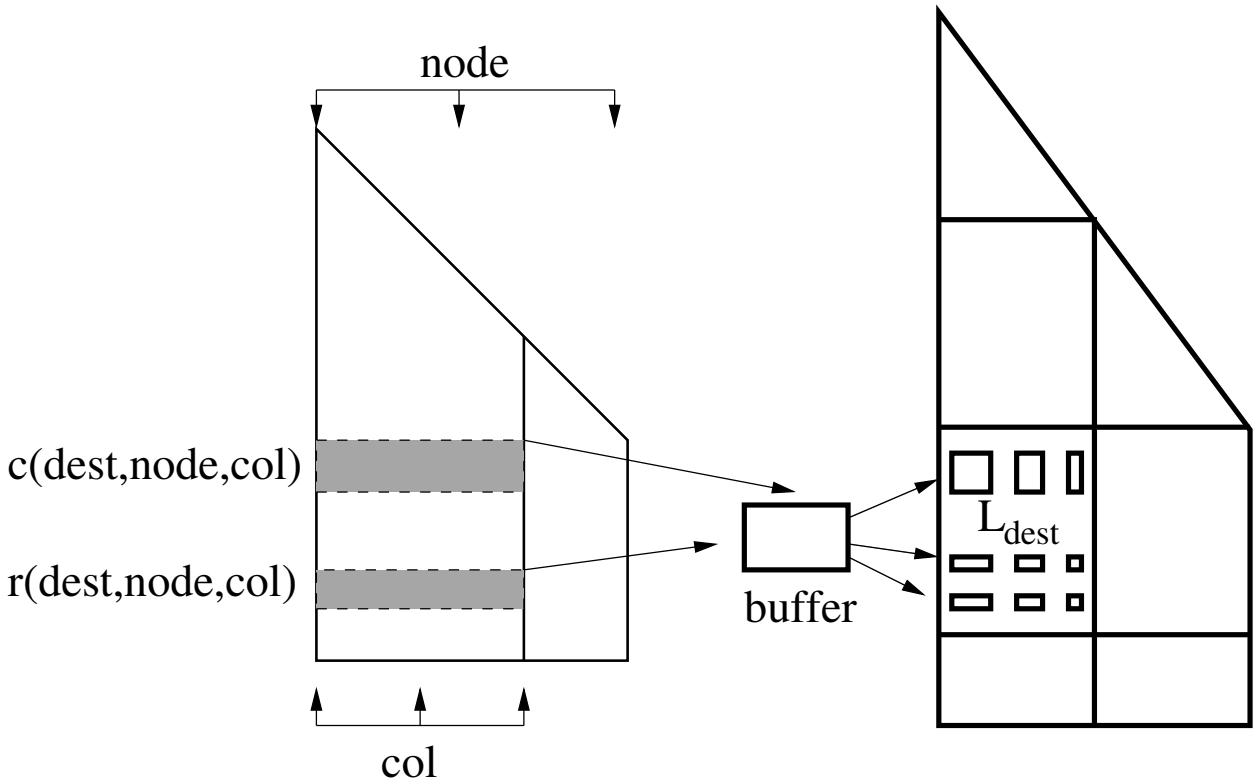
Tasks in sparse DAG

update_between: performs update between nodal matrices ie.

$$L_{dest} \Leftarrow L_{dest} - L_r L_c^T$$

where L_{dest} belongs to one nodal matrix and L_r and L_c belong to another.

update_between



1. Form outer product $L_r L_c^T$ into buffer.
2. Distribute results into destination block L_{dest}.

Dependency count

During analyse, calculate number of tasks to be performed for each block of L.

Dependency count

During analyse, calculate number of tasks to be performed for each block of L.

During factorization, keep running count of outstanding tasks for each block.

Dependency count

During analyse, calculate number of tasks to be performed for each block of L.

During factorization, keep running count of outstanding tasks for each block.

When count reaches 0 for block on the diagonal, release factorize task and decrement count for each off-diagonal block in its block column by one.

Dependency count

During analyse, calculate number of tasks to be performed for each block of L.

During factorization, keep running count of outstanding tasks for each block.

When count reaches 0 for block on the diagonal, release factorize task and decrement count for each off-diagonal block in its block column by one.

When count reaches 0 for off-diagonal block, release solve task and decrement count for blocks awaiting the solve by one. Update tasks may then be spawned.

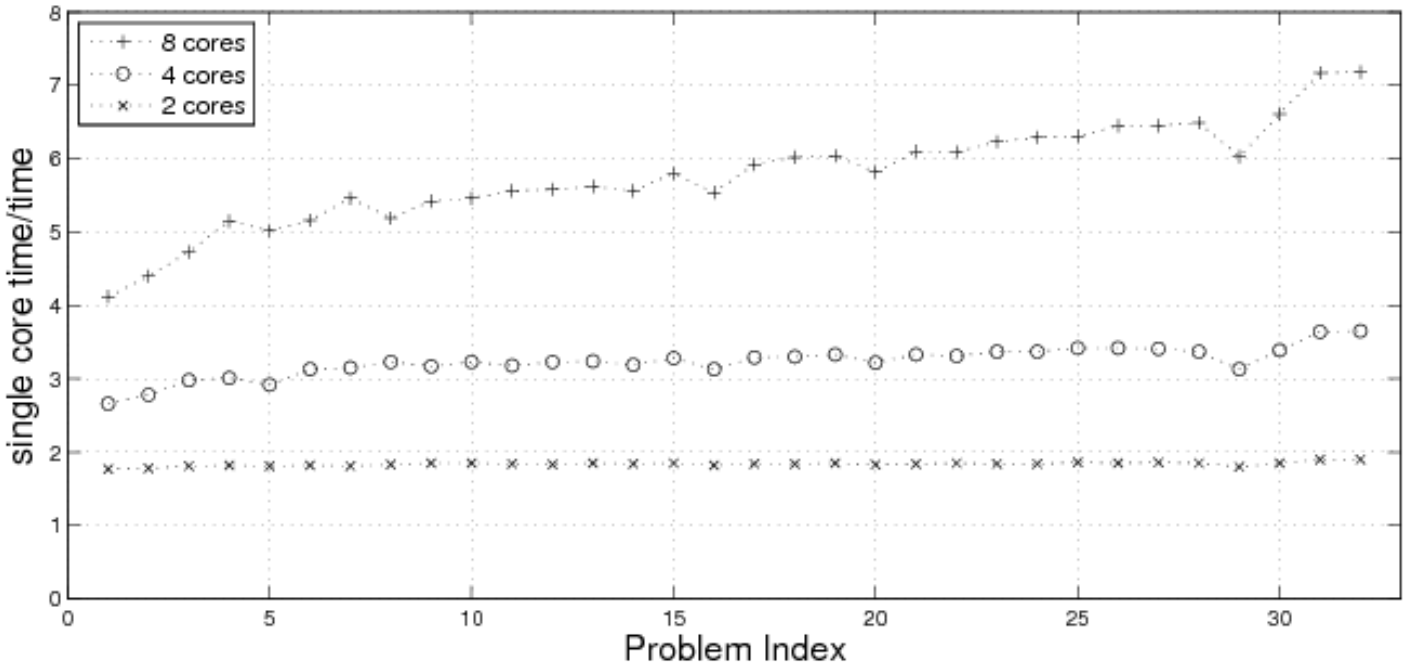
Task pool

Each cache keeps small stack of tasks that are intended for use by threads sharing this cache.

Tasks added to or drawn from top of local stack. If becomes full, move bottom half to task pool.

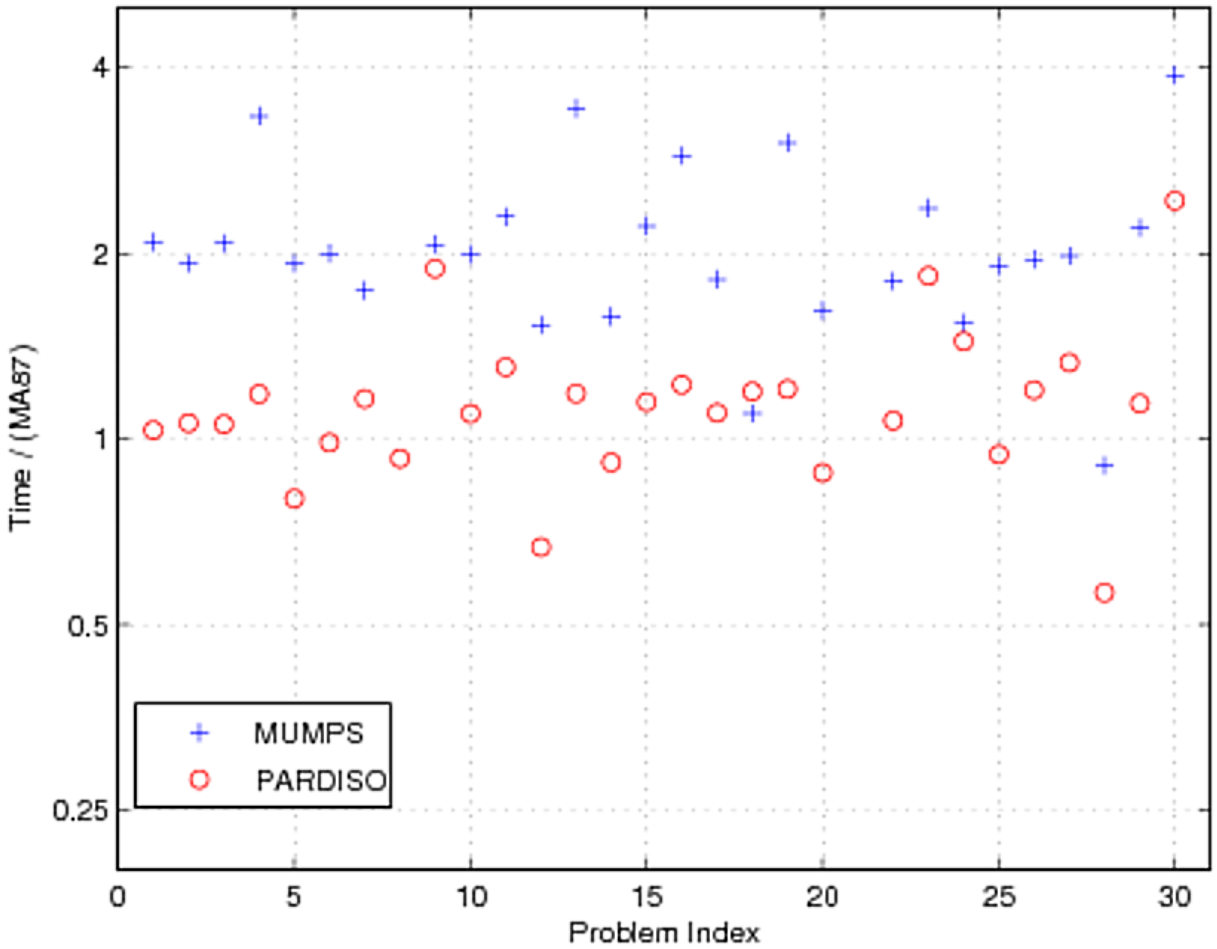
Sparse positive-definite DAG results

New sparse Cholesky solver is HSL_MA87.
The ratios of HSL_MA87 factorize times on 2, 4 and 8 cores to its factorize time on a single core.



Science & Technology Facilities Council
Rutherford Appleton Laboratory

Comparisons with other solvers on 8 threads



Science & Technology Facilities Council
Rutherford Appleton Laboratory

Indefinite case

Sparse DAG approach very encouraging for our 8 core machine

Indefinite case

Sparse DAG approach very encouraging for our 8 core machine

BUT

- So far, only considered positive definite case.
- Saddle-point systems are common eg in optimization.

$$\begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

Such systems are indefinite and Cholesky factorization $A = LL^T$ does not exist.

Indefinite case

In positive definite case, factor task was

$$A_k = L_{kk} L_{kk}^T.$$

In indefinite case, becomes

$$A_k = P_k L_{kk} D_k L_{kk}^T P_k^T,$$

where P_k is a permutation and D_k is (block) diagonal.

This involves pivoting (permuting large entries to the diagonal or sub-diagonal positions).

1×1 and 2×2 pivots are used.

Implications

- Searching for pivots involves all entries in block column.
- The factor and solve tasks on the block column must be combined.
- Thus less scope for parallelism (parallelism less fine-grained).
- A block column can only be factorized once its dependency count reaches zero.

Further implications

- Data structures from analyse have to be modified to accommodate delayed pivots (those that fail stability test).
- This results in more data movement/copying ... adds overhead.
- Code is even more complicated!

Our indefinite sparse solver is HSL_MA86.

HSL_MA86 results for positive-definite problems

Factorize times for running positive-definite problems without and with pivoting.

	No pivoting		Pivoting	
Problem	1	8	1	8
Boeing/bcsstk38	0.069	0.168	0.087	0.144
Simon/olafu	0.202	0.174	0.244	0.152
ND/nd6k	18.3	3.02	20.6	3.94
ND/nd12k	80.0	12.3	88.5	15.2

Note: smaller block used for indefinite case and this benefits smaller problems.

HSL_MA86 results: good news

Factorize times for indefinite problems on 1 and 8 cores.

Problem	1	8	speedup
Boeing/crystk03	1.29	0.36	3.58
Koutsovasilis/F2	2.57	0.57	4.51
Cunningham/qa8fk	4.23	0.88	4.79
Oberwolfach/t3dh	12.1	2.17	5.58
Schenk_AFE/af_shell10	72.8	11.7	6.22
Oberwolfach/bone010	590	88.3	6.68
PARSEC/Ga41As41H72	7290	1141	6.39

Conclude: very good results for some large problems

HSL_MA86 results: tough problems

Many delayed pivots cause performance hit.

Problem	num_delay	1	8	speedup
GHS_indef/sparsine	16	250	44.4	5.65
Schenk_IBMA/c-62	28728	9.07	4.93	1.84
GHS_indef/aug3d	144955	36.5	25.9	1.41

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Concluding remarks

- Extended DAG approach from dense positive-definite systems to sparse systems
- Very good results for factorizing positive-definite matrices on our 8-core machine
- Also good results for large indefinite problems provided there are few delayed pivots
- For some tough indefinite problems, further work needed to improve performance while maintaining stability.
- Our DAG-based solvers are available as part of HSL 2011.

Thank you!

Reports available:
`http://www.cse.scitech.ac.uk/nag/reports.shtml`