A DAG-based sparse Cholesky solver for multicore architectures

Jonathan Hogg
John Reid
Jennifer Scott

CSC09 Monterey Bay October 2009

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Outline of talk

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

- Introduction
- Dense systems
- Sparse systems
- Future directions and conclusions

Today A is positive definite.

Solving systems in parallel

Haven't we been solving linear systems in parallel for years?
Yes — large problems on distributed memory machines

We want to solve

- Medium and large problems (more than 10^{10} flops)
- On desktop machines
- Shared memory, complex cache-based architectures
- 2–8 cores now in all new machines.
- Soon 16–64 cores will be standard.

Traditional MPI methods work, but can we do better?

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Faster

I have an 8-core machine...

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

The dense problem

Solve

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

with A

- Symmetric and dense
- Positive definite (indefinite problems require pivoting)
- Not small (order at least a few hundred)

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Pen and paper approach

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

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

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 $A_{(k+1:n)(k+1:n)} \leftarrow A_{(k+1:n)(k+1:n)} - L_{(k+1:n)k}L^T_{(k+1:n)k}$

Science & Technology Facilities Council Rutherford Appleton Laboratory

Serial approach

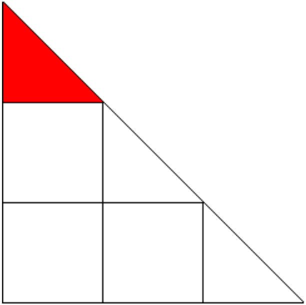
Exploit caches

Use algorithm by blocks

- Same algorithm, but submatrices not elements
- $10\times$ faster than a naive implementation
- Built using Basic Linear Algebra Subroutines (BLAS)

Cholesky by blocks

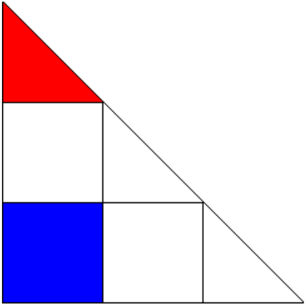
Factorize diagonal



Factor(col)

Cholesky by blocks

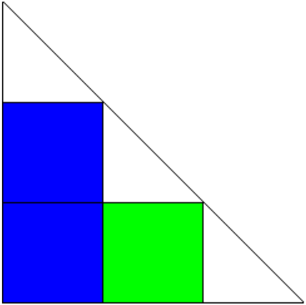
Solve column block



Solve(row, col)

Cholesky by blocks

Update block



Update(row, source col, target col)

Parallelism mechanisms

MPI	Designed for distributed memory, requires substantial changes
OpenMP	Designed for shared memory
pthreads	POSIX threads, no Fortran API
ITBB	Intel Thread Building Blocks, no Fortran API
Coarrays	Not yet widely supported

Parallelism mechanisms

MPI	Designed for distributed memory, requires substantial changes
OpenMP	Designed for shared memory
~~pthreads~~	~~POSIX threads, no Fortran API~~
~~ITBB~~	~~Intel Thread Building Blocks, no Fortran API~~
~~Coarrays~~	~~Not yet widely supported~~

Parallelism mechanisms

MPI	Designed for distributed memory, requires substantial changes
OpenMP	Designed for shared memory
~~pthreads~~	~~POSIX threads, no Fortran API~~
~~ITBB~~	~~Intel Thread Building Blocks, no Fortran API~~
~~Coarrays~~	~~Not yet widely supported~~

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Traditional approach

Just parallelise the operations

Solve(row,col) Can do the solve in parallel

Update(row,scol,tcol) Easily split as well

Traditional approach

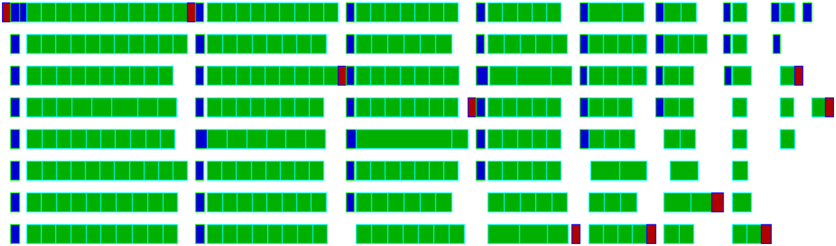
Just parallelise the operations

Solve(row,col) Can do the solve in parallel

Update(row,scol,tcol) Easily split as well

What does this look like...

Parallel right looking



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).

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

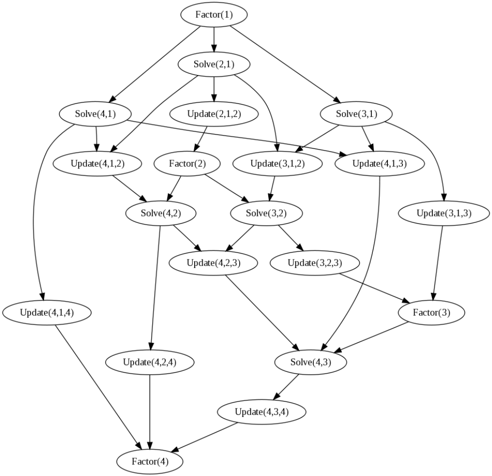
- Tasks are vertices
- Dependencies are directed edges

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

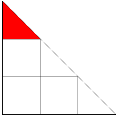
Approach used by Buttari, Dongarra, Kurzak, Langou, Luszczek, Tomov (2006)

Science & Technology Facilities Council
Rutherford Appleton Laboratory

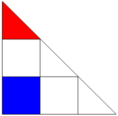
Task DAG



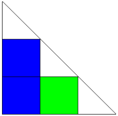
Factor(col) $A_{kk} = L_{kk}L_{kk}^T$



Solve(row, col) $L_{ik} = A_{ik}L_{kk}^{-T}$

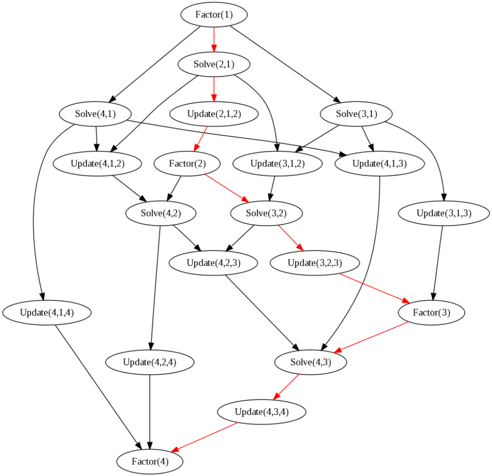


Update(row, scol, tcol)
$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

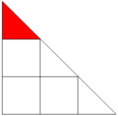


Science & Technology Facilities Council
Rutherford Appleton Laboratory

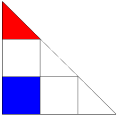
Task DAG



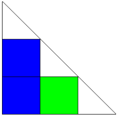
Factor(col) $A_{kk} = L_{kk} L_{kk}^T$



Solve(row, col) $L_{ik} = A_{ik} L_{kk}^{-T}$

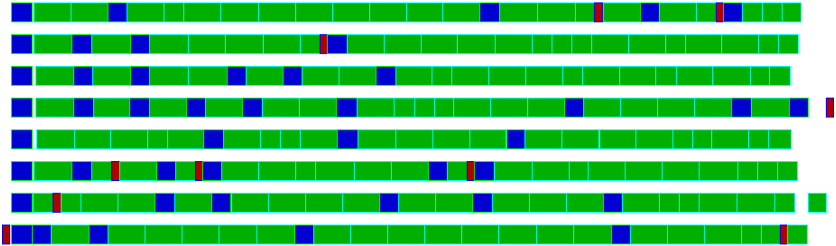


Update(row, scol, tcol)
$A_{ij} \leftarrow A_{ij} - L_{ik} L_{jk}^T$



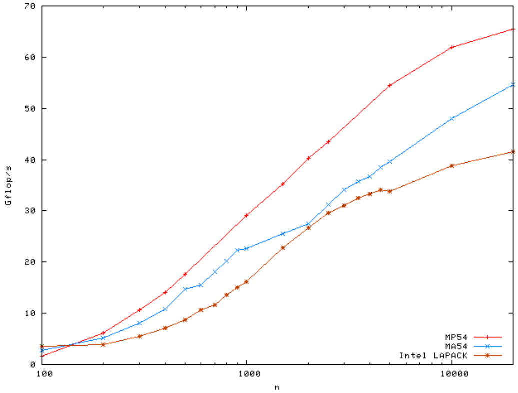
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Profile



Results

Performance using 8 threads (`dgemm` peak is 72.8 Gflop/s)



Speedup for dense case

n	Speedup
500	3.2
2500	5.7
10000	7.2
20000	7.4

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Speedup for dense case

n	Speedup
500	3.2
2500	5.7
10000	7.2
20000	7.4

New dense DAG code `HSL_MP54` available in `HSL2007`.

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.

Direct methods

Generally comprise four phases:

- **Reorder** Symmetric permutation P to reduce fill-in.
- **Analyse** Predict non-zero pattern. Build elimination tree.
- **Factorize** Using data structures built in analyse phase, perform the numerical factorization.
- **Solve** Using computed factors solve $A\mathbf{x} = \mathbf{b}$.

Aim: Organise computations to use dense kernels on submatrices.

Elimination and assembly tree

The elimination tree provides partial ordering of the operations.

If U is a descendant of V, we must factorize U first.

To exploit BLAS, combine adjacent nodes whose cols have same (or similar) sparsity structure.

Condensed tree is assembly tree.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Factorize phase

Existing parallel approaches usually rely on two levels of parallelism

Tree-level parallelism: assembly tree specifies only partial ordering (parent processed after its children). Independent subtrees processed in parallel.

Node-level parallelism: parallelism within operations at a node. Normally used near the root.

Factorize phase

Existing parallel approaches usually rely on two levels of parallelism

Tree-level parallelism: assembly tree specifies only partial ordering (parent processed after its children). Independent subtrees processed in parallel.

Node-level parallelism: parallelism within operations at a node. Normally used near the root.

Our experience: speedups less than ideal on multicore machines.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

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.

Hold set of contiguous cols of L with (nearly) same pattern as a dense trapezoidal matrix, referred to as nodal matrix.

Sparse DAG

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

Hold set of contiguous cols of L with (nearly) same pattern as a dense trapezoidal matrix, referred to as nodal matrix.

Divide the nodal matrix into blocks and perform tasks on the blocks.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Tasks in sparse DAG

factorize(`diag`) Computes dense Cholesky factor L_{triang} of the triangular part of block `diag` on diagonal. If block trapezoidal, perform triangular solve of rectangular part

$$L_{rect} \Leftarrow L_{rect} L_{triang}^{-T}$$

Tasks in sparse DAG

factorize(`diag`) Computes dense Cholesky factor L_{triang} of the triangular part of block `diag` on diagonal. If block trapezoidal, perform triangular solve of rectangular part

$$L_{rect} \Leftarrow L_{rect} L_{triang}^{-T}$$

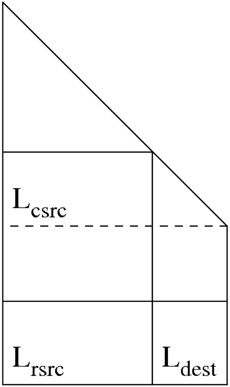
solve(`dest, diag`) Performs triangular solve of off-diagonal block `dest` by Cholesky factor L_{triang} of block `diag` on its diagonal.

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

Tasks in sparse DAG

update_internal(dest, rsrc, csrc)

Within nodal matrix, performs update



$$L_{dest} \Leftarrow L_{dest} - L_{rsrc} L_{csrc}^T$$

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Tasks in sparse DAG

`update_between(dest, snode, scol)`

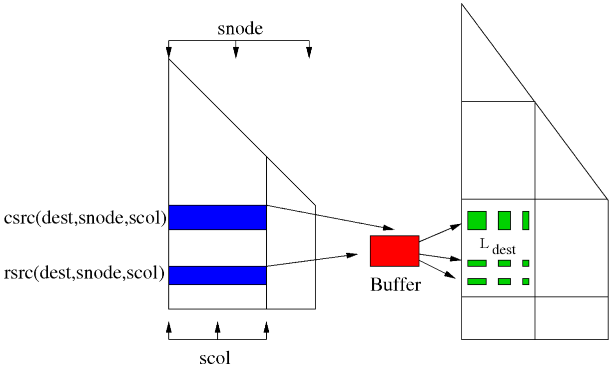
Performs update

$$L_{dest} \Leftarrow L_{dest} - L_{rsrc} L_{csrc}^T$$

- where L_{dest} is a submatrix of the block `dest` of an ancestor of node `snode`
- L_{rsrc} and L_{csrc} are submatrices of contiguous rows of block column `scol` of `snode`.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

update_between(dest, snode, scol)



1. Form outer product $L_{rsrc}L_{csrc}^T$ into Buffer.
2. Distribute the results into the destination block L_{dest}.

Dependency count

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

Science & Technology Facilities Council
Rutherford Appleton Laboratory

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, store factorize task and decrement count for each off-diagonal block in its block column by one.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

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, store factorize task and decrement count for each off-diagonal block in its block column by one.

When count reaches 0 for off-diagonal block, store 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.

Tasks in pool given priorities:

1.	**factorize**	Highest priority
2.	**solve**	
3.	**update_internal**	
4.	**update_between**	Lowest priority

Science & Technology Facilities Council
Rutherford Appleton Laboratory

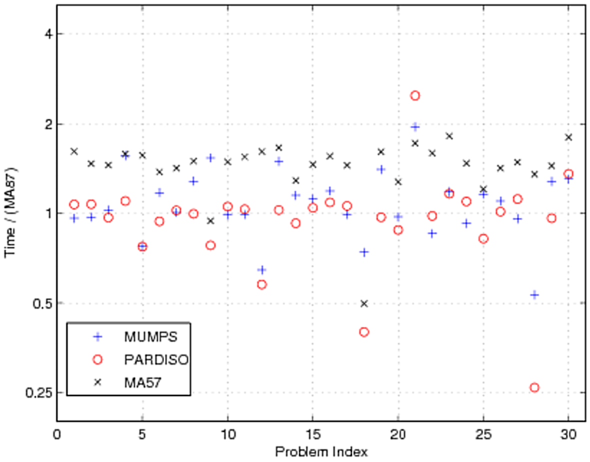
Sparse DAG results

Results on machine with 2 Intel E5420 quad core processors.

Problem	Time		Speedup
cores	1	8	
DNVS/thread	5.25	0.98	5.36
GHS_psdef/apache2	30.1	5.07	5.94
Koutsovasilis/F1	37.8	6.05	6.24
JGD_Trefethen/Trefethen_20000b	102	16.5	6.18
ND/nd24k	335	53.7	6.23

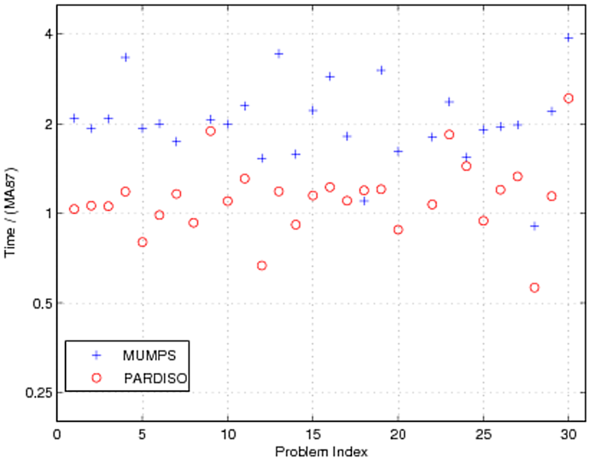
Science & Technology Facilities Council
Rutherford Appleton Laboratory

Comparisons with other solvers, one thread



Science & Technology Facilities Council
Rutherford Appleton Laboratory

Comparisons with other solvers, 8 threads



Indefinite case

Sparse DAG approach very encouraging for multicore architectures.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Indefinite case

Sparse DAG approach very encouraging for multicore architectures.

BUT

- Results reported so far, only for positive definite case.
- Indefinite case is harder because of pivoting.
- We use block column dependency counts and combine factor and solve tasks.
- Preliminary results: speed ups not quite so good.

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Code availability

New sparse DAG code is `HSL_MA87`.

To be included within HSL.

If you want to try it out, let us know.

Science & Technology Facilities Council
Rutherford Appleton Laboratory