Sparse Direct Solvers for Extreme-Scale Computing

Iain Duff

Joint work with Florent Lopez and Jonathan Hogg

STFC Rutherford Appleton Laboratory

SIAM Conference on Computational Science and Engineering (CSE17).
Hilton Atlanta, Atlanta, Georgia, USA. 27 February - 3 March 2017.

NLAFET
Parallel Numerical Linear Algebra for Future Extreme Scale Systems

- H2020 FET-HPC Project 671633
- Funding of around 4M Euros
- Partners are
  - Umeå University, Sweden .. Coordinator: Bo Kågström
  - University of Manchester, UK .. Jack Dongarra
  - INRIA, Paris, France .. Laura Grigori
  - STFC, UK .. Iain Duff
- Started 1 November 2015 (effectively Jan 2016)
- 36 months project terminating on 31 October 2018

NLAFET Work Packages

- WP1: Management and coordination
- WP2: Dense linear systems and eigenvalue problem solvers
- WP3: Direct solution of sparse linear systems
- WP4: Communication-optimal algorithms for iterative methods
- WP5: Challenging applications– a selection
  Material science, power systems, study of energy solutions, and data analysis in astrophysics
- WP6: Cross-cutting issues
  Scheduling and runtime systems, auto-tuning, fault tolerance
- WP7: Dissemination and community outreach

NLAFET .. Workpackage 3

- T3.1 Lower Bounds on Communication for Sparse Matrices
- T3.2 Direct Methods for (Near-)Symmetric Sparse Systems
- T3.3 Direct Methods for Highly Unsymmetric Sparse Systems
- T3.4 Hybrid Direct-Iterative Methods

Solution of sparse linear systems

We wish to solve the sparse linear system

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

where the sparse matrix $\mathbf{A}$ is sparse and of large dimension, typically 10^6 or greater, and we want to solve the system on an extreme scale computer.

Direct methods

We will consider the factorization:

$$\mathbf{P_r A P_c} \rightarrow \mathbf{LU}$$

L : Lower triangular (sparse)
U : Upper triangular (sparse)

Permutations $\mathbf{P_r}$ and $\mathbf{P_c}$ chosen to preserve sparsity and maintain stability

When $\mathbf{A}$ is symmetric $\mathbf{U} = \mathbf{DL^T}$ and $\mathbf{P_c} = \mathbf{P_r^T}$

Sparse direct methods

- Black boxes available

Sparse direct methods

- Black boxes available
- Complexity can be low. Almost linear storage in 2D

Sparse direct methods

- Black boxes available
- Complexity can be low. Almost linear storage in 2D
- Routinely solving problems of order in millions

Sparse direct methods

- Black boxes available
- Complexity can be low. Almost linear storage in 2D
- Routinely solving problems of order in millions
- There can be issues with storage requirement

Sparse direct methods

- Black boxes available
- Complexity can be low. Almost linear storage in 2D
- Routinely solving problems of order in millions
- There can be issues with storage requirement
- Target is half asymptotic speed of machine

Task 3.2 Direct Methods for (Near-)Symmetric Systems

Sparse LL^T, LDL^T, LU

- Tree-based solvers
- Use DAGs with runtime scheduling systems (WP6)

Runtime systems

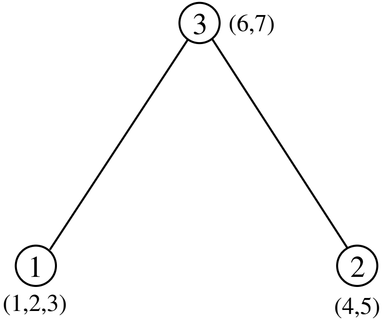
- Within the framework of NLAFET, we are primarily concerned with the runtime systems
  - StarPU using an STF (sequential task flow) model, and
  - PaRSEC using PTG (parametrized task graph) model
  - OpenMP Version 4.0 or above, using task features
- In all cases, we are using a task-based approach and the structure involved is a directed acyclic graph (DAG)

Sparse factorization

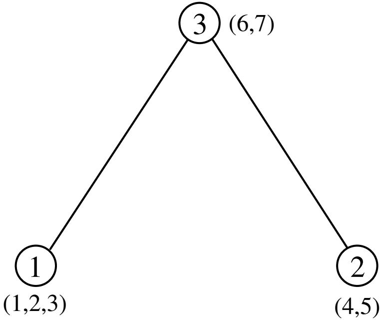
The kernel of most sparse direct codes is a dense factorization.

We feel it is useful to show this via a mini-tutorial

$$\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}$$



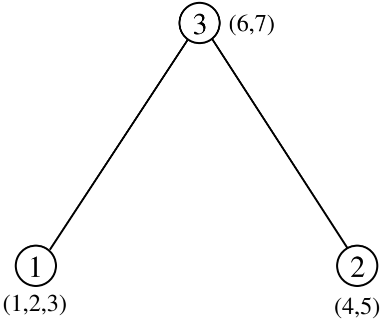
$$
\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}
$$



At step 1

$$
\begin{array}{c|ccc|cc}
 & 1 & 2 & 3 & 6 & 7 \\
\hline
1 & \times & \times & \times & \times & \\
2 & \times & \times & \times & \times & \times \\
3 & \times & \times & \times & \times & \times \\
\hline
6 & \times & \times & \times & \times & \times \\
7 & & \times & \times & \times & \times
\end{array}
$$

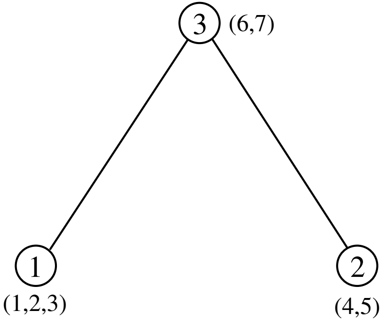
$$
\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}
$$



At step 2

$$
\begin{array}{cccc}
 & 4 & 5 & 6 \\
4 & \times & \times & \times \\
5 & \times & \times & \times \\
6 & \times & \times & \times
\end{array}
$$

$$\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}$$

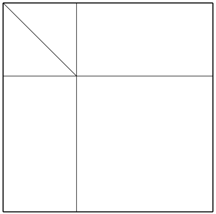


At step 3

$$\begin{array}{ccc} & 6 & 7 \\ 6 & \times & \times \\ 7 & \times & \times \end{array} + \begin{array}{ccc} & 6 & 7 \\ 6 & \color{red}{\times} & \color{red}{\times} \\ 7 & \color{red}{\times} & \color{red}{\times} \end{array} + \begin{array}{cc} & 6 \\ 6 & \color{brown}{\times} \end{array} \longrightarrow \begin{array}{ccc} & 6 & 7 \\ 6 & \times & \times \\ 7 & \times & \times \end{array}$$

Computation at node

The computation at a node involves dense factorization. Pivots are chosen from the top left block in the picture below but elimination operations are performed on the whole frontal matrix. Rows and columns of the factors can be stored and the resulting Schur complement (in bottom right) is passed up the tree for future assemblies.



Sparse parallelism

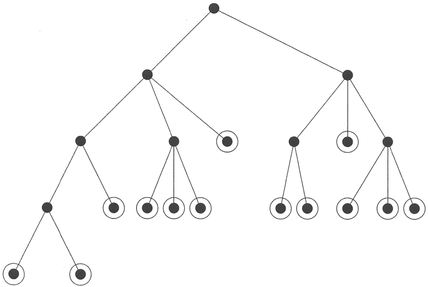
Sparse parallelism

- There are several levels of parallelism in sparse systems

Sparse parallelism

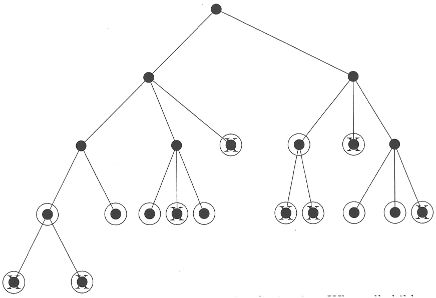
- There are several levels of parallelism in sparse systems
  - Partitioning ... block diagonal or block triangular form
  - Tree level parallelism
  - Node parallelism (including multi-threaded BLAS)
  - Inter-node parallelism

Tree parallelism



Available nodes at start of factorization. Work corresponding to leaf nodes can proceed immediately and independently.

Tree parallelism



Situation part way through the elimination. When all children of a node complete then work can commence at parent node.

Node and tree parallelism

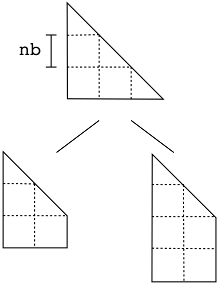
Matrix	Order	Tree nodes	Leaf nodes No.	Leaf nodes Av. size	Top 3 levels No.	Top 3 levels Av. size	Top 3 levels % ops
bratu3d	27 792	12 663	11 132	8	296	37	56
cont-300	180 895	90 429	74 673	6	10	846	41
cvxqp3	17 500	8 336	6 967	4	48	194	70
mario001	38 434	15 480	8 520	4	10	131	25
ncvxqp7	87 500	41 714	34 847	4	91	323	61
bmw3_2	227 362	14 095	5 758	50	11	1 919	44

Statistics on front sizes in assembly tree. From Duff, Erisman, Reid (2016).

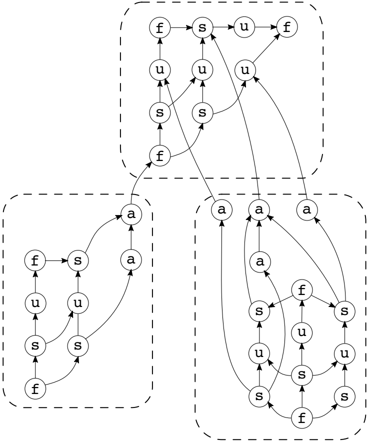
Node and tree parallelism

- Near the root there is not much tree parallelism but the nodes are large, that is the dense matrices at these nodes are of large dimension and so there is plenty of node parallelism.
- Conversely, near the leaves, there is little node parallelism but plenty of tree parallelism.

Inter-node parallelism

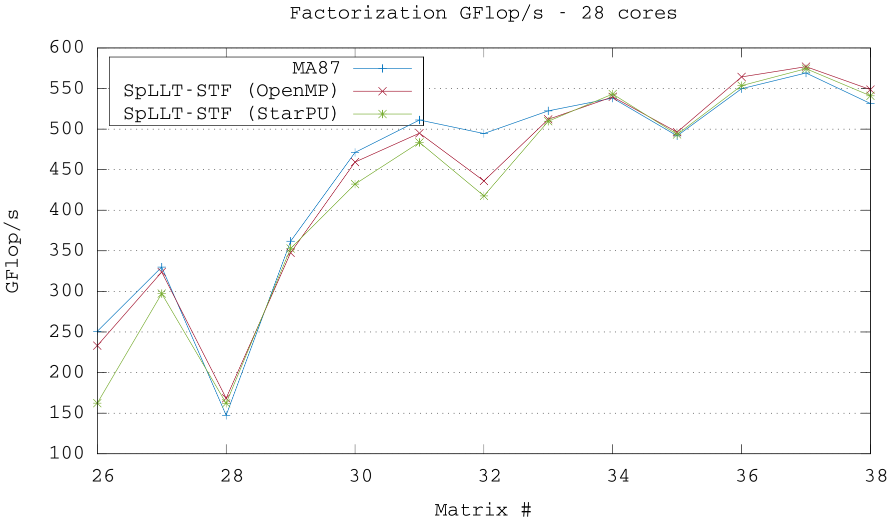


Part of tree



Directed acyclic graph

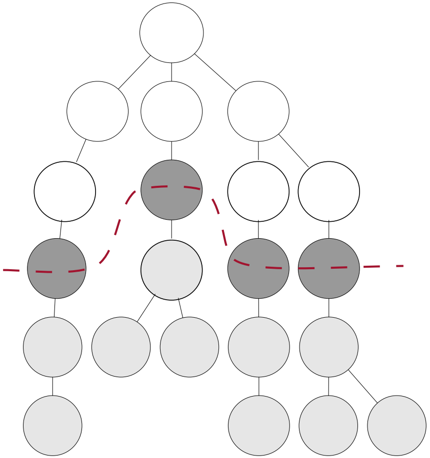
Sparse Cholesky using runtime systems



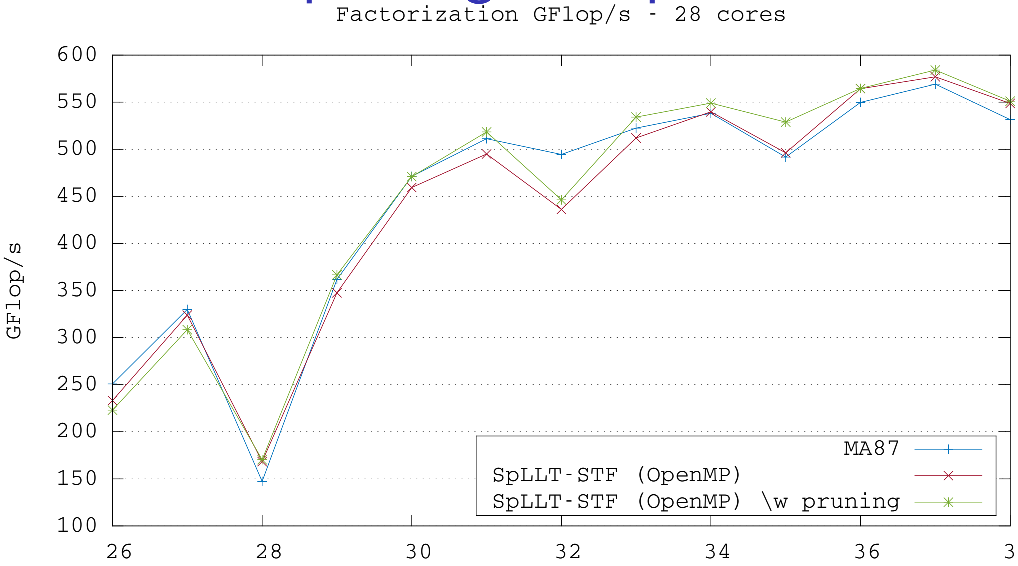
The main problem with using the runtime systems (for example matrix 15) is that when the tasks are small, the overhead in setting them up in the runtime system predominates.

We can avoid some of this by grouping together into a single task nodes near the leaves of the tree. We call this tree pruning.

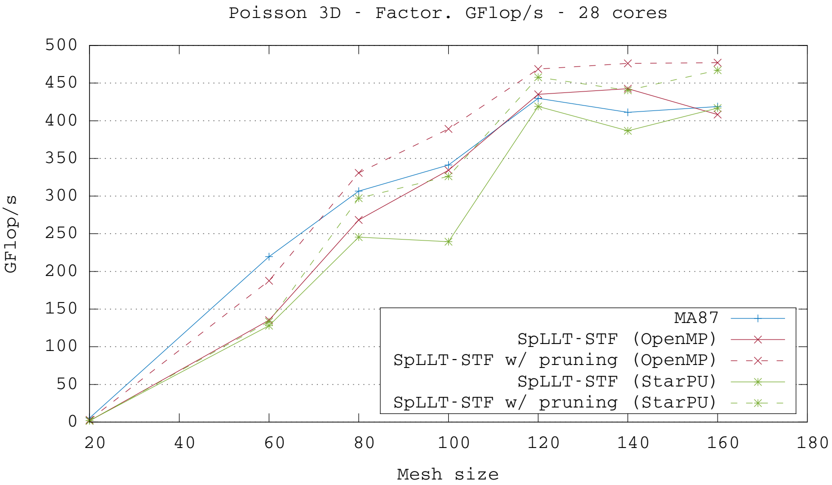
Tree pruning strategy



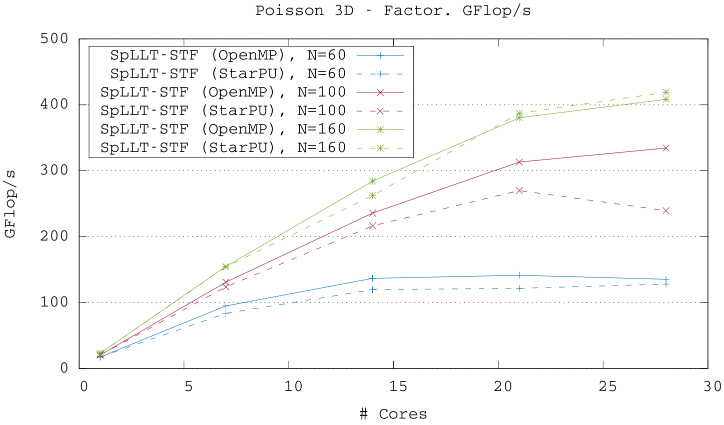
Effect of tree pruning on OpenMP version



Runs on large three-dimensional problems



Scalability



Symmetric indefinite matrices

If the matrix is indefinite then numerical pivoting is needed.

A simple example is the matrix

$$\begin{bmatrix} 0 & \times \\ \times & 0 \end{bmatrix}$$

Numerical pivoting in indefinite case

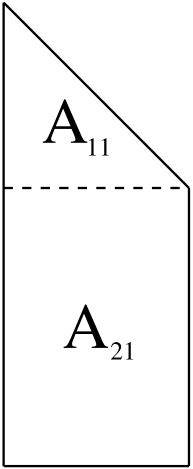
Good news is that we can stably factorize an indefinite matrix using only 1×1 and 2×2 pivots (Bunch and Kaufmann).

As is standard in sparse factorization, we use threshold rather than partial pivoting so we want ...

$$\|Pivot\| \geq u \times \|Largest\ in\ column\|$$

where u is the threshold parameter $(0 < u \leq 1)$.

Numerical pivoting in indefinite case

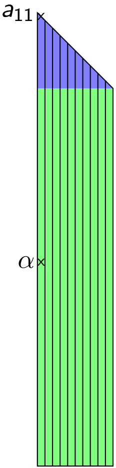


Pivots can only be chosen from A_{11}.

Can restrict pivoting to A_{11} or only choose pivots from A_{11} but then fail if large entry in A_{21} causes pivot to fail test.

Is a real problem when implementing algorithm on parallel machine.

Threshold Partial Pivoting



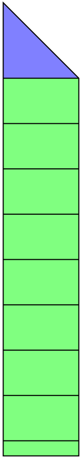
TPP Algorithm

- Work column by column
- Bring column up-to-date
- Find maximum element α in column of A_{21}
- Pivot test $\alpha / a_{11} < u^{-1}$. Accept/reject pivot

Problems

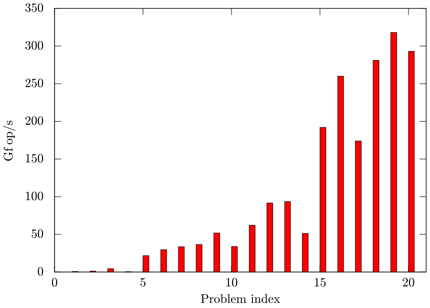
- Very stop-start (one column at a time)
- Communication for every column

A Posteriori Pivoting



- Work by blocks of A_{21}
- Every block uses factors of A_{11}
- Every block checks $\max |l_{21}| < u^{-1}$
- Communication when all blocks are done
- Discard all columns to right of failed entry
- Scaling and ordering $\Rightarrow$ failed pivots are *rare*

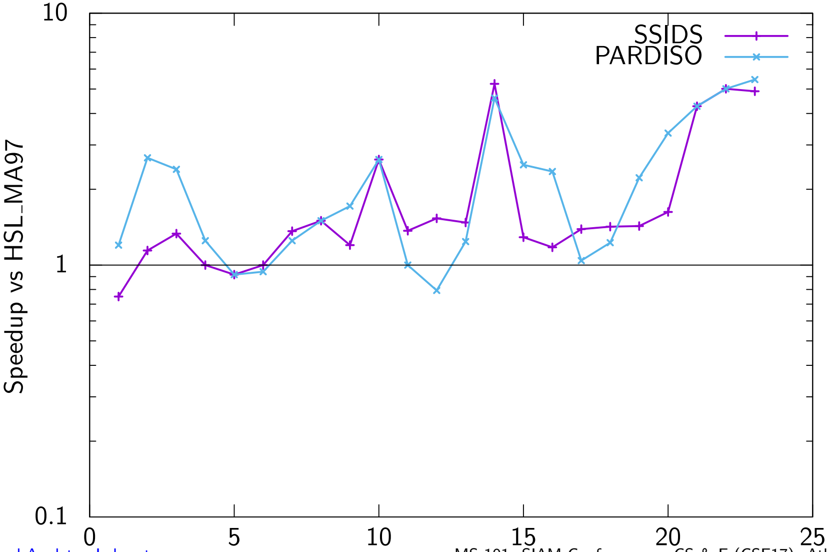
Flop rate



Machine achieves 794 Gflop/s on LINPACK

Compare new code SSIDS with standard TPP code HSL_MA97 and with PARDISO as implemented in MKL.

Hard indefinite on 28-core machine



Numerical pivoting in indefinite case

- HSL_MA97 Threshold partial pivoting
- PARDISO Only chooses pivots from A_{11}
- SSIDS Uses a posteriori pivoting

Numerical pivoting in indefinite case

- HSL_MA97 Threshold partial pivoting
- PARDISO Only chooses pivots from A_{11}
- SSIDS Uses a posteriori pivoting

I parodize these as

Numerical pivoting in indefinite case

- HSL_MA97 Threshold partial pivoting
- PARDISO Only chooses pivots from A_{11}
- SSIDS Uses a posteriori pivoting

I parodize these as

- HSL_MA97 PAY

Numerical pivoting in indefinite case

- HSL_MA97 Threshold partial pivoting
- PARDISO Only chooses pivots from A_{11}
- SSIDS Uses a posteriori pivoting

I parodize these as

- HSL_MA97 PAY
- PARDISO PRAY

Numerical pivoting in indefinite case

- HSL_MA97 Threshold partial pivoting
- PARDISO Only chooses pivots from A_{11}
- SSIDS Uses a posteriori pivoting

I parodize these as

- HSL_MA97 PAY
- PARDISO PRAY
- SSIDS PLAY

Hard indefinite on 28-core machine

Matrix	stokes128	cvxqp3	ncvxqp7
Order $\times 10^3$	49.7	17.5	87.5
Entries $\times 10^6$	0.30	0.07	0.31
	Factor time		
HSL_MA97	0.15	1.52	8.18
PARDISO	0.12	0.33	1.50
SSIDS V2	0.11	0.29	1.67
	Backward error		
HSL_MA97	$1.6\ 10^{-15}$	$3.1\ 10^{-11}$	$4.4\ 10^{-9}$
PARDISO	$3.9\ 10^{-3}$	$1.1\ 10^{-6}$	$1.4\ 10^{-7}$
SSIDS V2	$1.4\ 10^{-15}$	$2.0\ 10^{-11}$	$7.3\ 10^{-9}$

Summary

- There is lots of parallelism in sparse direct solvers

Summary

- There is lots of parallelism in sparse direct solvers
- Using a runtime system can compete with hand-coded versions

Summary

- There is lots of parallelism in sparse direct solvers
- Using a runtime system can compete with hand-coded versions
- Pivoting can be accommodated with little overhead in performance

Summary

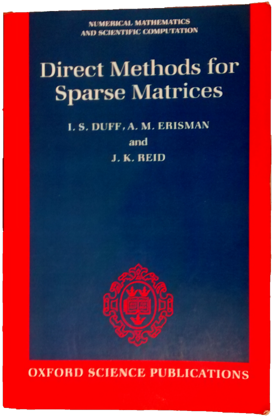
- There is lots of parallelism in sparse direct solvers
- Using a runtime system can compete with hand-coded versions
- Pivoting can be accommodated with little overhead in performance
- We meet our goal of running at half the peak performance

Summary

- There is lots of parallelism in sparse direct solvers
- Using a runtime system can compete with hand-coded versions
- Pivoting can be accommodated with little overhead in performance
- We meet our goal of running at half the peak performance
- Programming this is tough

THANK YOU FOR YOUR ATTENTION

Direct methods



Published by OUP in 1986

Direct methods