A Sparse Direct Solver for GPUs

Jonathan Hogg,
Evgueni Ovtchinnikov,
Jennifer Scott*

STFC Rutherford Appleton Laboratory

12 February 2014
Oxford e-Research Centre
Many-core Seminar

* Thanks also to Jeremy Appleyard of NVIDIA

Aims

Sparse $Ax = b$.
Fast.

Direct methods Factorize matrix $A = LU$ then triangular solves.

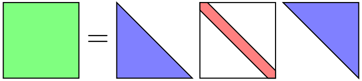
- MATLAB backslash easy.
- Black box - works 99.999% of the time
- GPU libraries: few/none

Iterative methods CG and friends.

- Expertise required to pick correct method
- Often requires preconditioning
- Doesn't work for all matrices
- GPU libraries: many

Factorization

Factorize as:



$$A = L \; D \; L^T$$

- Sparse
- Symmetric: $A = A^T$
- Non-singular (for simplicity!)

Modern direct solver design

Four phases

Ordering	Find fill-reducing permutation
Analyse	Find dense submatrix structure. Setup data representation.
Factor	Perform factorization with pivoting.
Solve	Use factorization to solve $Ax = b$.

Science & Technology Facilities Council

Modern direct solver design

Four phases

Ordering Find fill-reducing permutation

Analyse Find dense submatrix structure.
Setup data representation.

Factor Perform factorization with pivoting.

Solve Use factorization to solve $Ax = b$.

GPU Challenges

- Thousands of *small* dense subproblems (e.g. 8×1)
- Pivoting on *large* dense subproblems (e.g. 4000×2000)
- Substantial sparse scatter/gather
- Complicated kernels (register pressure)

Science & Technology Facilities Council

Previous work

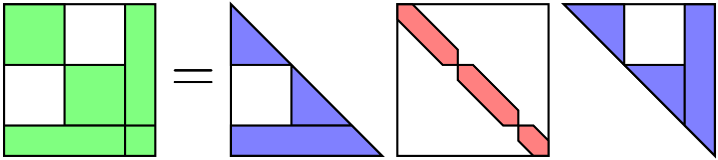
Pre-existing work

- Just offloading large BLAS 3/LAPACK operations.
  Very modest speedups on whole problem.
- A few codes go beyond this.
  None publicly available?
  No pivoting: potentially unstable
  Fairly modest speedups: CPU↔GPU bottleneck

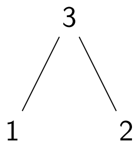
Our implementation

- Puts entire factorization and solve phases on GPU
- Open source, including all auxiliary codes
- Delivers over 5× speedup vs 2 CPU sockets on large problems

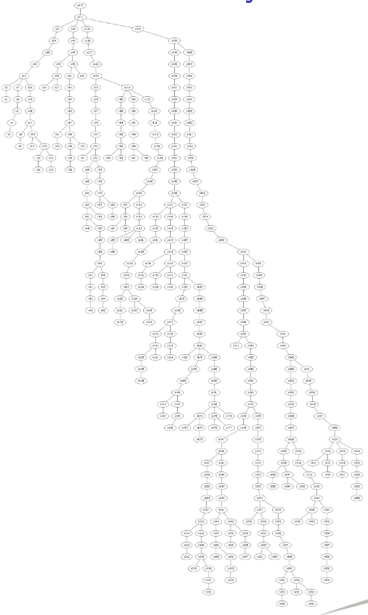
Tree parallelism



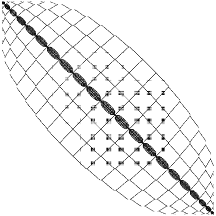
Operations in first two block columns are independent.
Data flow graph called Assembly Tree



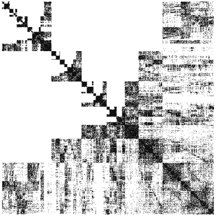
Real world assembly tree: PARSEC/SiNa



Original:



Reordered:



Node parallelism

For an individual block, <u>in order</u>:

Assemble	contributions from children (sparse gather)
Factor	$m \times k$ matrix with threshold pivoting (partial dense LDL^T)
Contribution	given by Schur complement (dgemm)

Each task itself can be parallelized (some better than others!)

First challenge: Exploit **both** tree and node parallelism

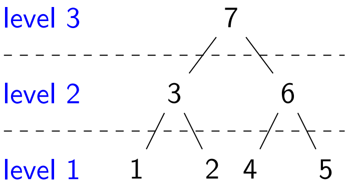
Note: CUBLAS only supports multiple BLAS on same dimensions.
⇒Have to write our own routines.

- CPU populates a data structure of tasks
- Assigns an appropriate number of blocks to each task
- Launches a kernel on $\sum$ blocks
- Costs several registers to do this (can't use constant cache)

Enforcing task ordering

Need to enforce assembly tree ordering

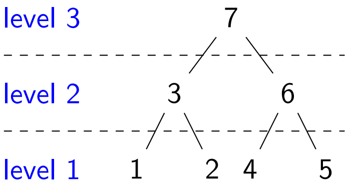
- Ideally would do so via global memory with single kernel
- Want to support Fermi, insufficient registers
- Use level based approach instead



Enforcing task ordering

Need to enforce assembly tree ordering

- Ideally would do so via global memory with single kernel
- Want to support Fermi, insufficient registers
- Use level based approach instead

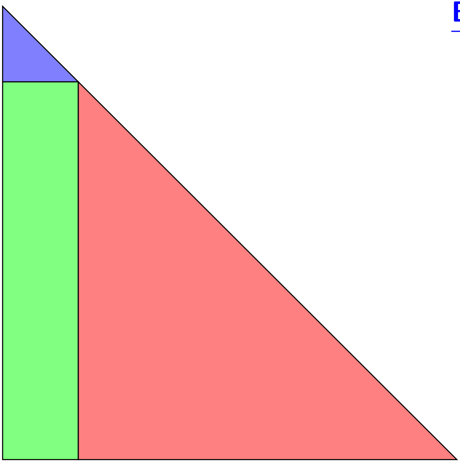


Outstanding Issues

Load balance:

- Disparate node sizes
- Freedom of assignment

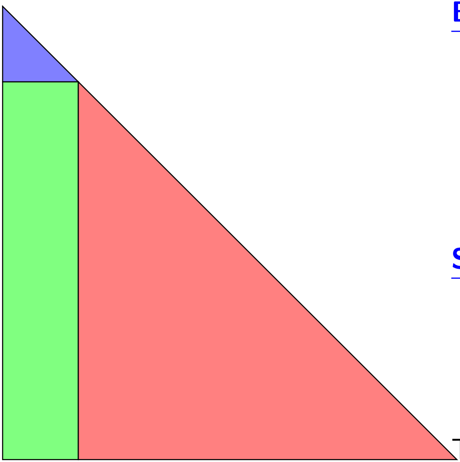
Factorization: basics



Basic Algorithm

1. Factor $A_{11} = L_{11} D_1 L_{11}^T$
2. Divide $L_{21} = A_{21} L_{11}^{-T}$
3. Form $C = L_{21} D_1 L_{21}^T$

Factorization: basics



Basic Algorithm

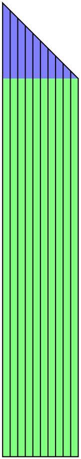
1. Factor $A_{11} = L_{11} D_1 L_{11}^T$
2. Divide $L_{21} = A_{21} L_{11}^{-T}$
3. Form $C = L_{21} D_1 L_{21}^T$

Stability

- **All** entries in $L_{21} < u^{-1}$
- Entries of D_1 calculated in stable fashion

Typically $u = 0.01$.

Factorization: parallel pivoting I

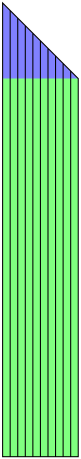


Traditional algorithm

- 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

Science & Technology Facilities Council

Factorization: parallel pivoting I



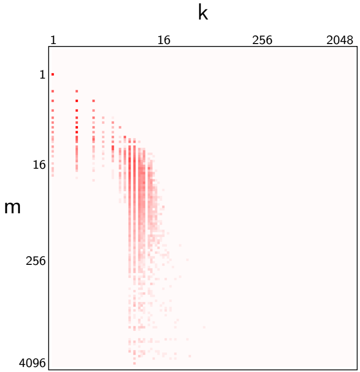
Traditional algorithm

- 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

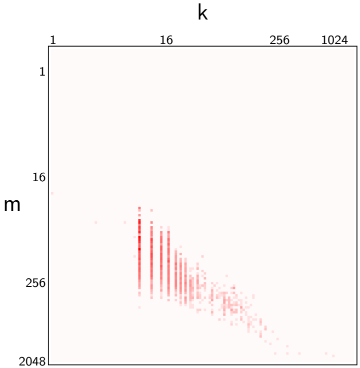
Problems

- Very stop-start (one column at a time)
- All-to-all communication for every column

Size distributions



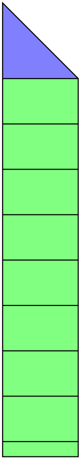
Schenk_IBMNA/c-big



GHS_psdef/bmwcra_1

- Wide range of sizes
- Often $m \gg k$

Factorization: parallel pivoting II



Solution

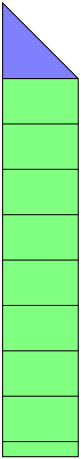
- Try-it-and-see pivoting (*a posteriori pivoting*)

New algorithm

- Work by blocks of L_{21}
- Every block factorizes copy of A_{11}
- Every block checks $\max |l_{21}| < u^{-1}$
- All-to-all communication when all blocks are done
- Discard columns that have failed on *any* block

We use a block size of 32×8.

Factorization: parallel pivoting III

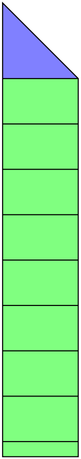


Implementation Issues

- Inefficient if lots of rejected pivots
- Still quite stop-start
- High register pressure (especially on Fermi)

Science & Technology Facilities Council

Factorization: parallel pivoting III



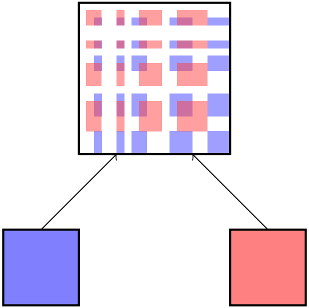
Implementation Issues

- Inefficient if lots of rejected pivots
- Still quite stop-start
- High register pressure (especially on Fermi)

Future work

- Implement Subset pivoting or other CA technique as fall back
- Move to DAG-based implementation (Kepler only) (Significant performance improvement expected)

Assembly: Sparse gather/scatter



Can be framed as *either* sparse gather *or* sparse scatter.

- Need to enforce ordering: prefer sparse gather
- Launch one kernel per child
  (i.e. all first children, then all second, ...)

Science & Technology Facilities Council

Auxiliary codes

Many auxiliary routines are required that are still CPU-based:

- Ordering (Nested Dissection)
- Analyse (Assorted Graph Algorithms)
- Scaling (MC64 or SpMv)

... but only run once for a sequence of problems

Auction-based scaling: alternative to MC64

For some problems, serial MC64 scaling takes $> 75\%$ of time

- 95% of the quality
- 10% of the time
- Parallelizable

Science & Technology Facilities Council

Results

Comparison

- C2050 GPU (Fermi) [515GFlops, 238 TDP]
- $2\times$ Xeon E5620 = 8 cores (Westmere-EP) [76.8GFlops, 160W TDP]
- Flops ratio about $7\times$

Test Problems

- $4\times$ Optimization (IPM)
- $4\times$ Finite Element
- $4\times$ Finite Difference

Times(s) and Speedup: Factor+Solve

Problem	CPU	GPU	Speedup
GHS_indef/c-72	0.48	0.35	1.37
GHS_indef/c-71	2.98	0.64	4.66
GHS_indef/ncvxqp3	10.65	2.03	5.25
Schenk_IBMNA/c-big	12.37	2.64	4.69
Nasa/nasasrb	0.88	0.17	5.18
DNVS/shipsec1	4.18	0.90	4.64
GHS_psdef/bmwcra_1	4.45	0.93	4.78
DNVS/ship_003	9.52	2.16	4.41
McRae/ecology1	1.64	0.94	1.75
AMD/G3_circuit	4.54	2.13	2.13
GHS_psdef/apache2	11.50	2.64	4.36
Lin/Lin	17.89	2.97	6.02

Science & Technology Facilities Council

Code hot-spots

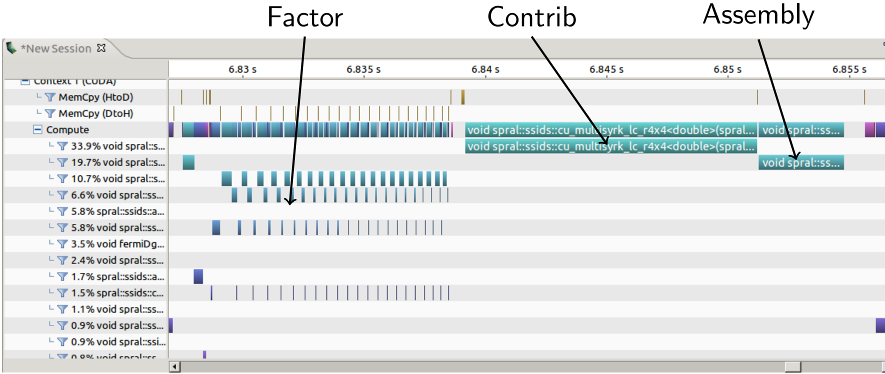
	c-72	c-big	shipsec1	Lin
Speedup	1.37	4.69	2.33	6.02
Contrib	19	780	1607	1568
Assembly	27	446	38	302
Factor	82	481	850	666
Waiting	143	525	405	352

Times are in ms.

Waiting = time not in kernels.

Science & Technology Facilities Council

Factor is poor



Conclusions and Future Work

Story so far

- New open source sparse direct solver in CUDA
  - Will be released with a little more tidying
- Speedups over host of around 5 on large problems
- Needed to both:
  - Handle peculiarities of device
  - Use new algorithms for massive parallelism

Near Future

- Multi-GPU

Long-term

- DAG-based factor
- GPU-based scaling
- Auto-generation from stencil?

Science & Technology Facilities Council

Thanks for listening!

Questions?

A Supplementary slide

Some supplementary text.
(Note numbering of supplementary slides is outside that of normal slides.)