Science & Technology Facilities Council

DAG-Scheduled Linear Algebra Using Template-Based Building Blocks

Jonathan Hogg

STFC Rutherford Appleton Laboratory

19 March 2015
GPU Technology Conference
San Jose, California

* Thanks also to Jeremy Appleyard of NVIDIA

Introduction

What's in the title?

DAG-Scheduled Similar approach to MAGMA, but more flexible.

Linear Algebra Aimed at implementing matrix algorithms-by-blocks.

Template-Based Building-Blocks Template library for BLAS-like functionality (i.e. CUB for LA)

Introduction

What's in the title?

DAG-Scheduled Similar approach to MAGMA, but more flexible.

Linear Algebra Aimed at implementing matrix algorithms-by-blocks.

Template-Based Building-Blocks Template library for BLAS-like functionality (i.e. CUB for LA)

So what's different to MAGMA?

- DAG handled on-device.
- Improved performance for small and medium matrices
- More flexible $\Rightarrow$ allows more complex pivoting

Introduction

What's in the title?

DAG-Scheduled Similar approach to MAGMA, but more flexible.

Linear Algebra Aimed at implementing matrix algorithms-by-blocks.

Template-Based Building-Blocks Template library for BLAS-like functionality (i.e. CUB for LA)

So what's different to MAGMA?

- DAG handled on-device.
- Improved performance for small and medium matrices
- More flexible $\Rightarrow$ allows more complex pivoting

Some things worked, some didn't...

DAG-Scheduling: Overview

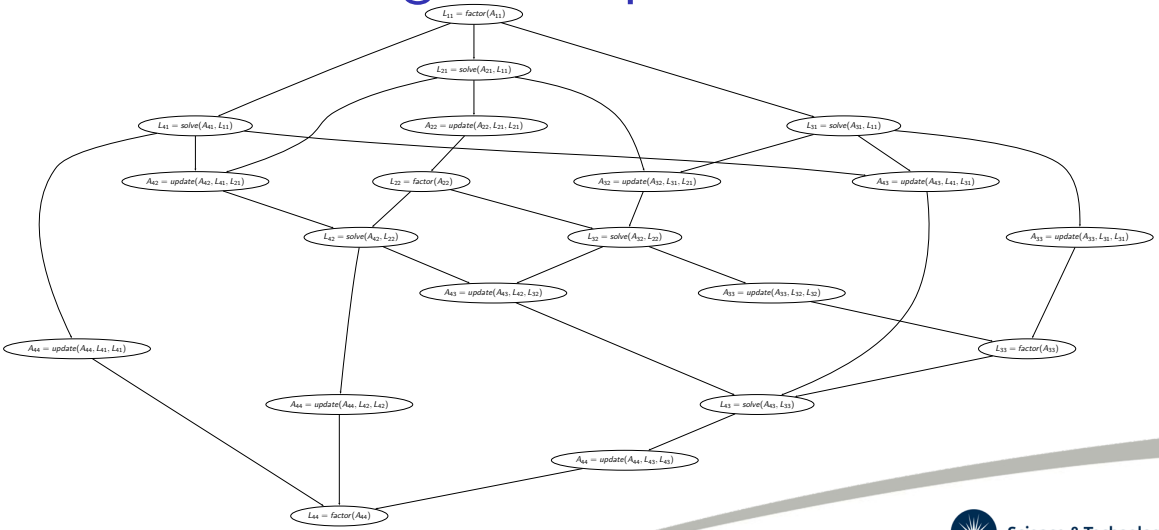
Aims

- Expose maximum parallelism
- Separate parallelism/scheduling from algorithm

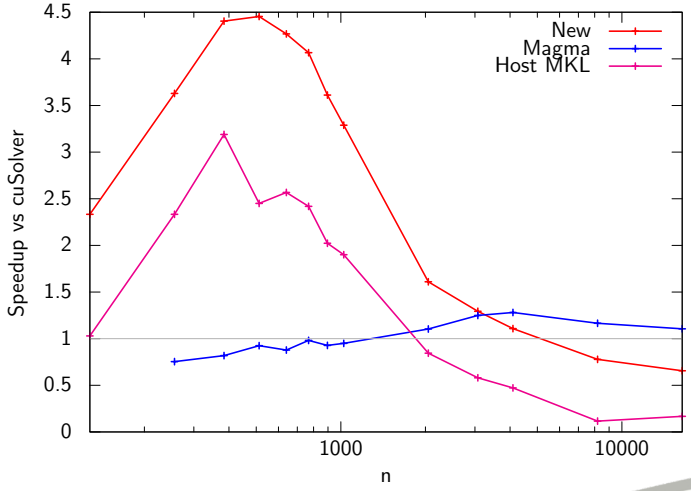
Example: Cholesky factorization

- Split matrix up into blocks
- Divide algorithm into tasks that act on blocks.
- Represent dependencies as edges in DAG
- Typically each task is implemented by a block of threads.

DAG-Scheduling: Example



DAG-Scheduling Progress: Cholesky



- More advanced implicit-DAG scheme similar to "domino" scheme from `_trsv`.
- Big gains on latency-bound sizes
- Still need to address flop-bound case by calling cuBLAS.
- Surprisingly beat MKL on "small" sizes

Science & Technology Facilities Council

Linear Algebra

Algorithms of interest

Cholesky $A = LL^T$ — proof of concept, check performance

Symmetric Indefinite $A = LDL^T$ — requires complex pivoting, (Bunch-Kaufmann often insufficient for sparse solvers).

Linear Algebra

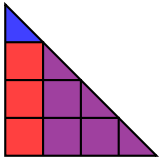
Algorithms of interest

Cholesky $A = LL^T$ — proof of concept, check performance

Symmetric Indefinite $A = LDL^T$ — requires complex pivoting, (Bunch-Kaufmann often insufficient for sparse solvers).

Cholesky

- For $j = 1, \ldots, n$:
  1. Factor diagonal block $L_{jj}L_{jj}^T \leftarrow A_{jj}$
  2. "Divide" column by diagonal $L_{ij} \leftarrow A_{ij}L_{jj}^{-T}, i > j$
  3. Update columns to right $A_{ik} \leftarrow A_{ik} - L_{ij}L_{kj}^T, i \geq k > j$



Symmetric Indefinite with Pivoting

Symmetric Indefinite $A = LDL^T$

- Ignoring stability, is essentially Cholesky with extra D's.
- To ensure stability need to ensure no entry of L is too large.
- For use in sparse solver, needs to cope with rectangular matrices
  $\Rightarrow$ Bunch-Kaufmann is unsuitable.

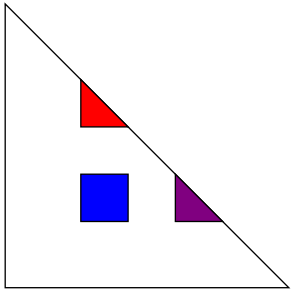
Traditional pivoting

- Finds largest entry in column before making pivoting decision.
- — Latency-bound (global communication for each column).
- — Entire (block) column may not fit in GPU (shared) memory.

Symmetric Indefinite with Pivoting II

But we're lucky!

- Numerical pre-treatment (scaling,ordering) means $< 0.1\%$ matrices need pivoting
- Allows "Try-it-and-see" approach (aka *A Posteriori Pivoting*)



Requirements from task system

- Follow Cholesky scheme

But also...

- Apply permutations to Left
- Check pivot sizes
- Perform speculative execution...
- ...backtrack if things go wrong
- In case where pivots fail, need to update to Left as well as Right

Requirements from task system

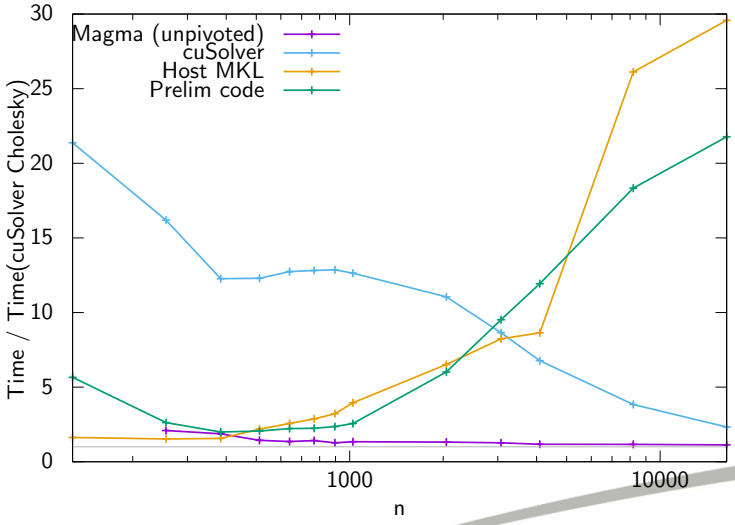
- Follow Cholesky scheme

But also...

- Apply permutations to Left
- Check pivot sizes
- Perform speculative execution...
- ...backtrack if things go wrong
- In case where pivots fail, need to update to Left as well as Right

Still writing this...
...but don't forsee major problems

Unoptimized results



DAG-Scheduling: Vs MAGMA

Implementation in MAGMA

+ Performs "straight-forward" tasks on GPU (e.g. _GEMM)
+ More complicated tasks on CPU (e.g. pivoting kernels)
+ High asymptotic performance (because _GEMM)

DAG-Scheduling: Vs MAGMA

Implementation in MAGMA

- \+ Performs "straight-forward" tasks on GPU (e.g. _GEMM)
- \+ More complicated tasks on CPU (e.g. pivoting kernels)
- \+ High asymptotic performance (because _GEMM)
- − Tasks must be certain minimum size to be efficient.
- − CPU↔GPU latency limits performance on small matrices.
- − Can't easily handle speculative execution and backtracking.
- − Doesn't work well on **lots** of simultaneous **small** matrices.
- − Can't (easily) dynamically modify task DAG based on pivoting decisions.

Template Library

What is it?

- Similar in concept to CUDA Unbound (CUB) library
- Provide efficient BLAS-like functionality as templates: "BLAS Unbound"
- Warp, Block and Device-level constructs
- Facilitate auto-tuning

Why do we need it?

- For our DAG-library, all tasks performed in same kernel
- So all get same shared memory, number of threads etc.
- Pick best parameters for `GEMM` operation where most flops are
- Everything else has to live within that envelope

The good, the bad and the ugly

Problems

- Combinatorial complexity / Manpower intensive
- Often need to break warp/block separation for performance
- Lots of performance optimization needed
- Can't even come close to cuBLAS GEMM performance (70% vs 90% of peak)

Wins

- Easy to play around with alternatives
- Test-driven development allows increased confidence in correctness
- Non-traditional features added using template parameters may be reused in other scenarios

Tricks for fast Cholesky

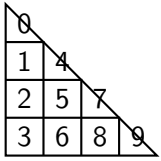
Warp-level

- Each thread handle multiple consecutive columns
- Hide DFMA in communication latency
- Can't hide in RSQRT latency — PTXAS issue?
- Use of SHFL requires a lot more unrolling — Instruction Cache Size issues
- Explicit hand/template based unrolling as NVCC tries to be too clever
- warpSize not a square number makes things messy
- Break block/warp separation by leaving $\frac{1}{\sqrt{d_{ii}}}$ on diagonal not $\sqrt{d_{ii}}$.

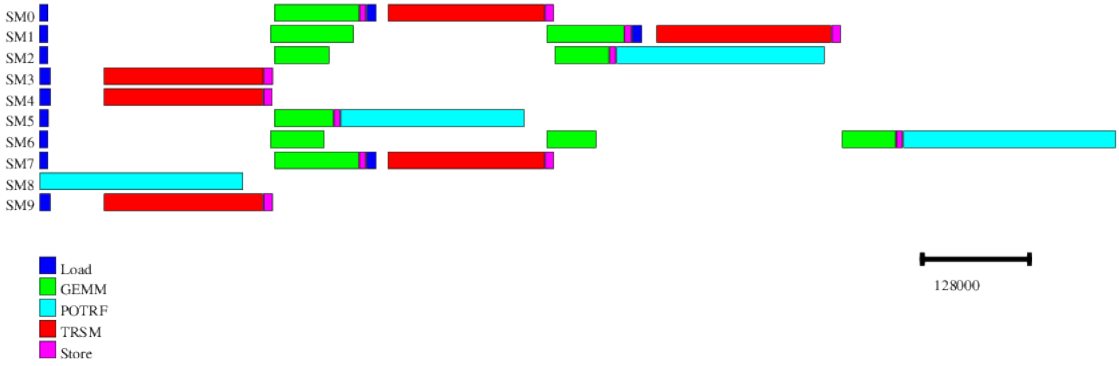
Tricks for fast Cholesky II

Block-level

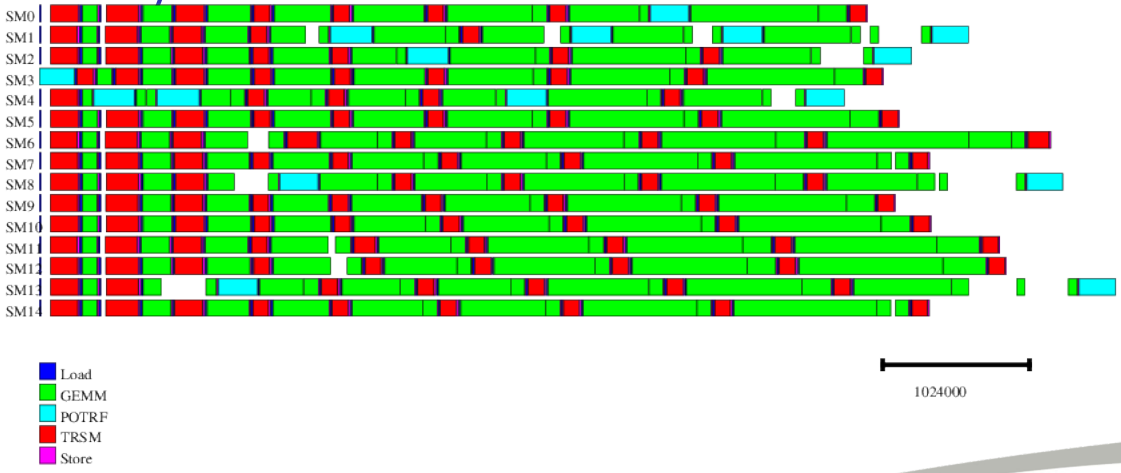
- Use different data layout for warps factoring diagonal vs off-diagonal
- Trsm (in Potrf): Each thread holds one entire vector $\Rightarrow$ no communication
- Work lower triangle of 4×4 blocks with only 8 warps — need "warp stealing".
- BlockTrsm: Stage D_{ii} into shmem by hand, double buffering
- BlockTrsm: Good SHFL use is fiddly
- Post progress after each diagonal block for device-level algorithm to pick up



Cholesky: 4 blocks



Cholesky: 16 blocks



Tricks for fast Cholesky III

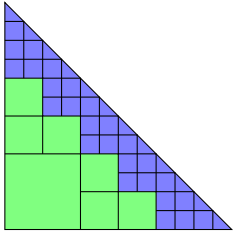
Device-level

- Overlap as much as possible
- Consolidate work and avoid synchronization
- Small calls to CUBLAS infeasible: launch overhead >> single block update
- Need to identify larger blocks to call CUBLAS on

Tricks for fast Cholesky III

Device-level

- Overlap as much as possible
- Consolidate work and avoid synchronization
- Small calls to CUBLAS infeasible: launch overhead >> single block update
- Need to identify larger blocks to call CUBLAS on
- Will need more complicated scheme:



Conclusions

- On-device DAG-scheduling good for latency bound kernels
- Significant improvements in Cholesky for $n \leq 2000$
- New a posteriori pivoting techniques for LDL^T
- BLAS/LAPACK-like template library is a lot of work
- ... so only limited subset will be brought up to release quality
- If you want the rest, email me.
- Code will ultimately be used for Sparse solver SPRAL SSIDS v2

jonathan.hogg@stfc.ac.uk

Thanks for listening!

Questions?

jonathan.hogg@stfc.ac.uk

`http://www.numerical.rl.ac.uk/spral`