Example of Cholesky’s Efficient Implementations

J. Kurzak
P. Luszczek
M. Gates
H. Anzt
**Scope**

batched spotrf

**precision**
- IEEE double precision
- IEEE single precision
- relaxed single precision

**size**
- large (up to 500)
- small (up to 100)
- ultra small (~20)
- variable size
- fixed size

**technology**
- composite kernel
- monolithic kernel

*machine learning*

*autotuning*
Motivation
machine learning

doubts
- Are they really so small?
- Is there really so many?
- Are they fixed size?

Alternating Least Squares
- Apache Mahout
- Spark MLlib
- GraphLab (Dato)
- Intel DAAL

Netflix Prize
- batch size: 17,000 and 500,000
- matrix size: 10 – 100
- uniform batch
Techniques

kernel development

coding

- C++ templates (parametrization)
- #pragma unroll (low level unrolling)
- pyexpander (high level unrolling)

algorithmic

- LAPACK-style blocking
- PLASMA-style tiling
- lazy evaluation (left / top looking)

Texture Reads

- texture objects
- __ldg() intrinsic

Vector Types

- double2, float4, ...

Tools

- nvprof
- nvdisasm

basically techniques for optimizing serial performance for memory efficiency and ILP
Thank you for your letters of collaboration!

- NVIDIA
- Intel
- AMD
Cholesky

**Sportf**

### Canonical
- BLAS 2
- Memory bound

### Blocking (LAPACK)
- Data locality
- Register reuse
- Surface to volume effect

### Lazy Evaluation
- Left-looking
- Memory efficiency
- Minimizing writes
Cholesky
spotrf

lazy evaluation / left-looking / “out of core”
- maximizes data reuse
- minimizes writes
Use right-looking algorithms to maximize SIMT parallelism.
Do wasteful work, but minimum number of conditionals.

Cholesky

Useful work

Actual work
For each matrix size $N$ tune:
- panel width (NB)
- thread block shape
  $(\text{blockDim.x}, \text{blockDim.y})$
- *not an exhaustive sweep*
Implementation and Tuning of Batched Cholesky Factorization and Solve for NVIDIA GPUs
IEEE Transactions on Parallel and Distributed Systems
http://dx.doi.org/10.1109/TPDS.2015.2481890
Batched Matrix Factorization

- cuBLAS batched LU
- BEAST tuned batched Cholesky

Matrix Dimension:
- 32: 8x FASTER
- 48: 15x FASTER
- 64: 12x FASTER
- 80: 12x FASTER
- 96: 11x FASTER

Time (ms):
- 0
- 20
- 40
- 60
- 80
- 100
- 120
- 140
- 160
- 180
Cholesky

spotrs

Solve

- single right hand side
- $L$ in lower triangle
- $L^T$ in upper triangle

- multiple solves in each thread block
Cholesky

`sposv`
ALS speedup over Spark

**Accelerating Collaborative Filtering Using Concepts from High Performance Computing**
2015 IEEE International Conference on Big Data
DOI: 10.1109/BigData.2015.7363811
Cholesky
one thread per matrix

Pros
- zero synchronization
- zero load imbalance
- no shared memory

Cons
- no cache / shared memory reuse
- unthinkable on standard layout
  
  *basically requires batch-major layout*

- LAPACKE: column-major / row-major
- cuDNN: NCHW / NHWC
Cholesky
one thread per matrix

algorithmic

- PLASMA-style tiling
- the laziest evaluation (top-looking)

basically completely serial implementation from the standpoint of each thread

no parallelization or vectorization considerations
tuning parameters

- right-looking, left-looking, top-looking
- thread block length (blockDim.x)
- tile size (NB)
- unrolling tile operations of the full factorization

Cholesky
one thread per matrix
Cholesky performance

Batched Cholesky on NVIDIA Kepler K40c

- **Interleaved, fast_math**
- **Interleaved**
- **cuBLAS LU**

Gflop/s vs. matrix dimension
Cholesky performance

Batched Cholesky on NVIDIA Kepler K40c

- **Interleaved, fast_math**
- **Interleaved**
- **Column-major, fast_math**
- **Column-major**
- **cuBLAS LU**

Gflop/s vs. matrix dimension
Conclusions

- For batched on GPUs you have to write specialized routines.
- We know how (common DLA wisdom applies).
- Autotuning works like a charm.

- For the most part on CPUs MKL+OpenMP gets the job done.

- Unorthodox layouts?
- Layout translation?
- On the fly?