High Performance Design of Batched Tensor Computations: Performance Analysis, Modeling, Tuning and Optimization

Azzam Haidar

Ahmad Abdelfattah, Jack Dongarra, Stan Tomov

MAGMA team @ ICL . UTK . EDU

MAGMA: Batched, Tensor, Deep Learning, Embedded, LA

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

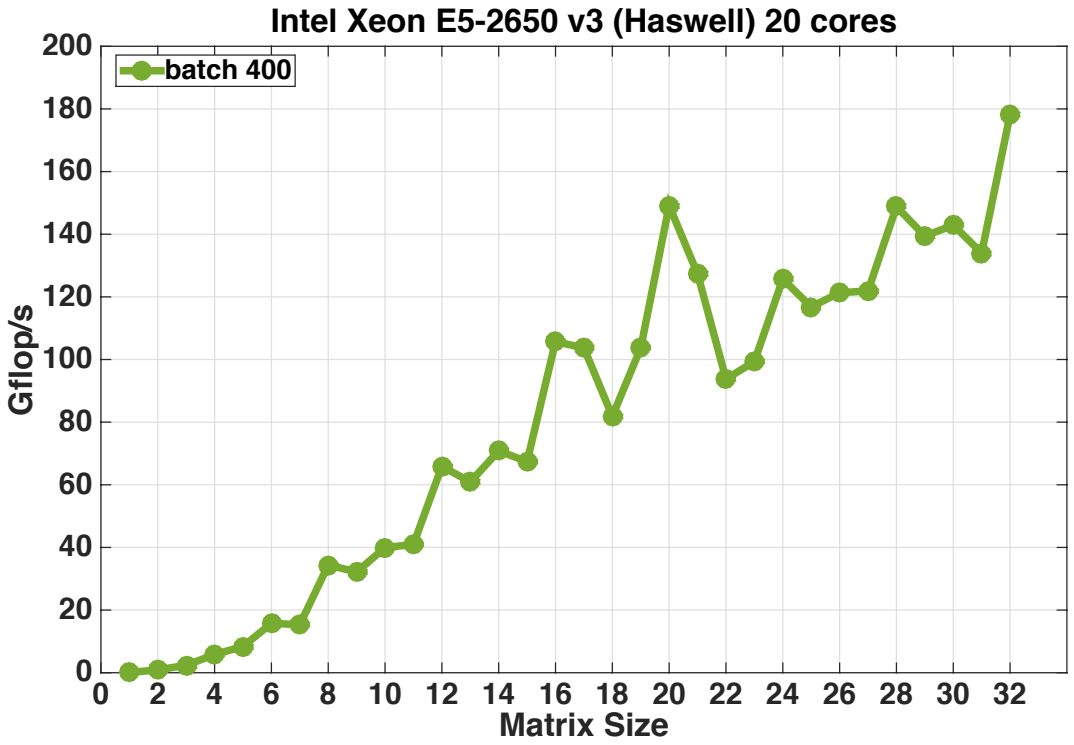
Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results

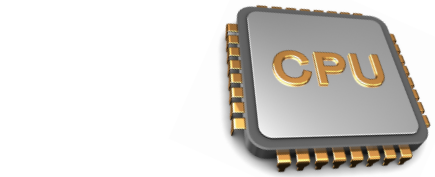
Reproducibility and reliability

- Working on MAGMA customized kernel for Deep Learning and tensor contraction
- How a 2X speedup can be faster than 10X
- Some of my observations on benchmark reporting
- I am not going to talk about accuracy reproducibility (Jim cover it) but rather I am going to talk benchmark reproducibility

Reproducibility and reliability




```
initialize (A, B, C);
start_timer
call mydgemm_batched
end_timer
```



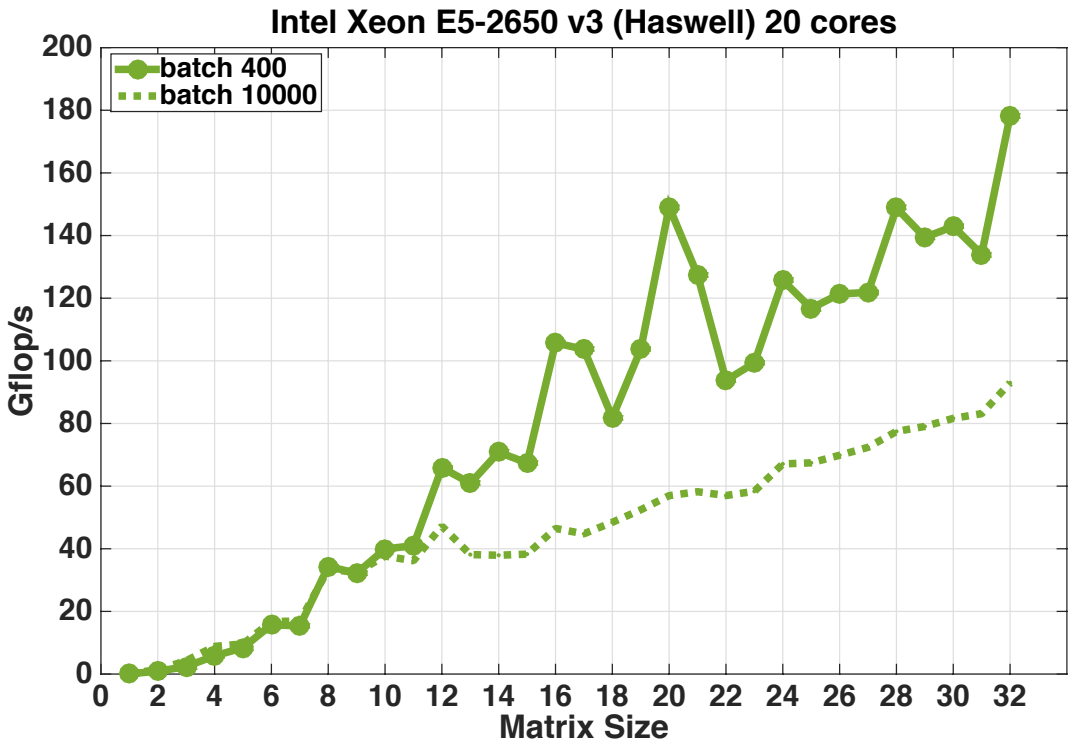
L2 cache

DRAM

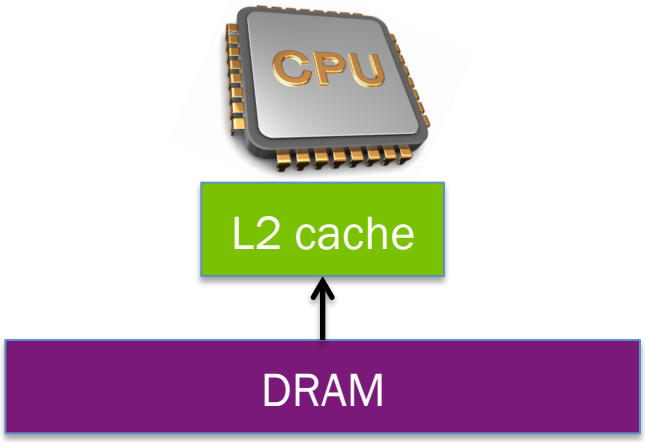
ICL

THE UNIVERSITY OF TENNESSEE KNOXVILLE

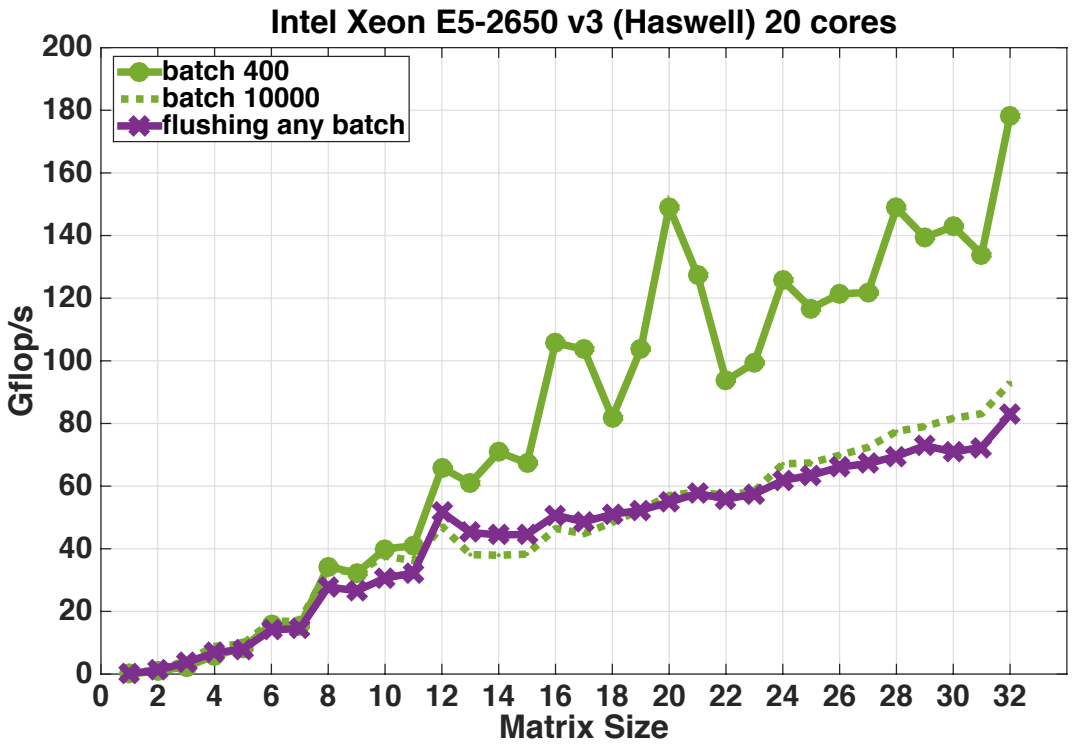
Reproducibility and reliability



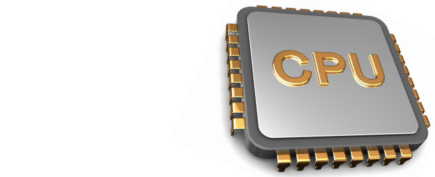

```
initialize (A, B, C);
start_timer
call mydgemm_batched
end_timer
```



Reproducibility and reliability



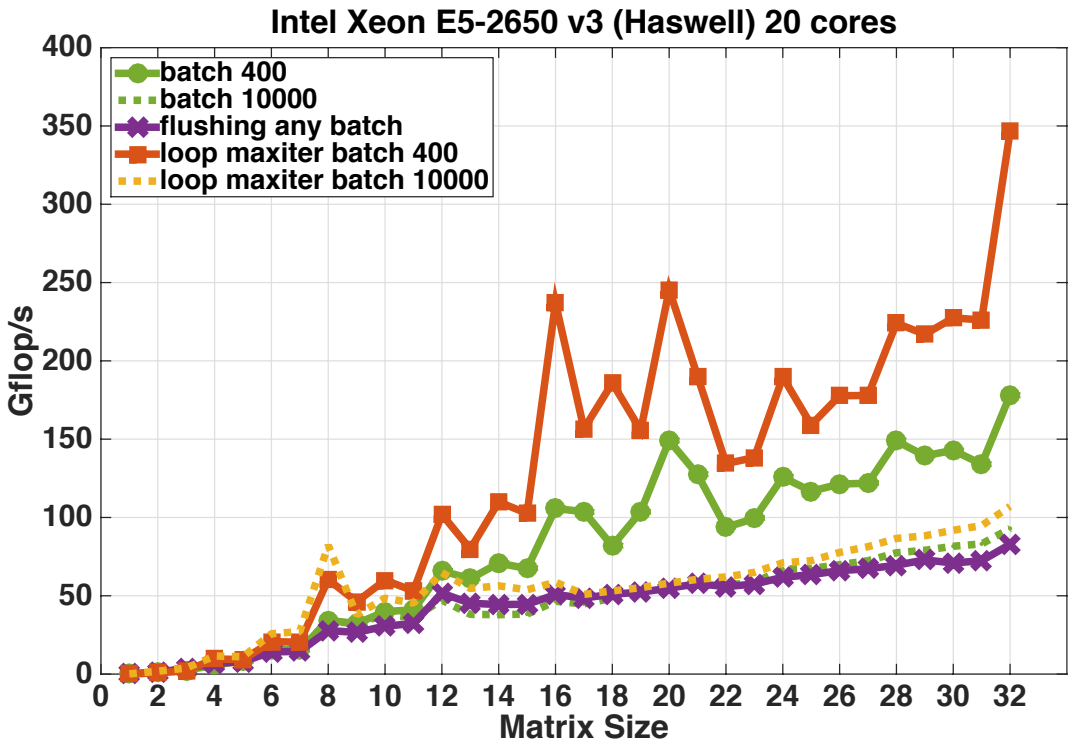

```
initialize (A, B, C);
flush cache large data
start_timer
call mydgemm_batched
end_timer
```



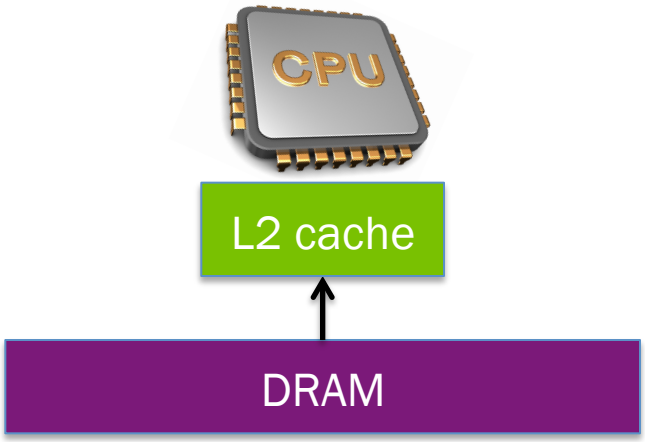
L2 cache

DRAM

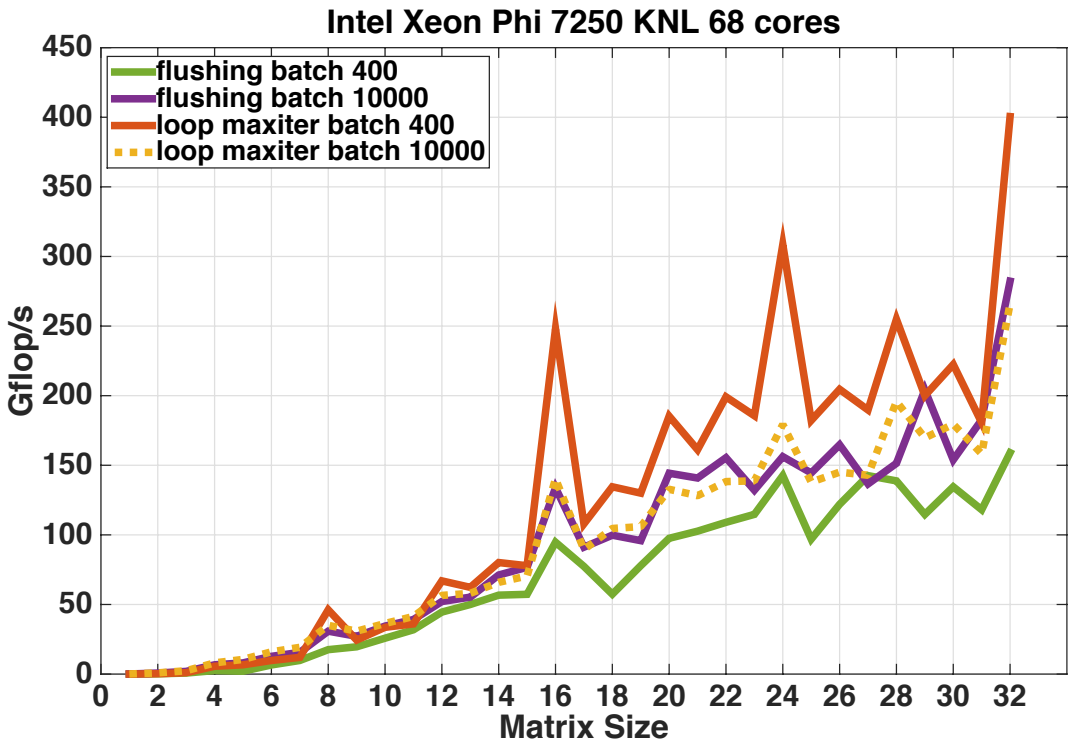
Reproducibility and reliability



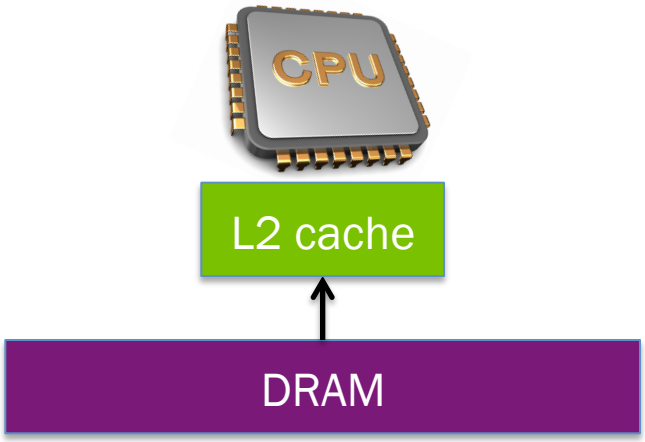

```
initialize (A, B, C);
flush cache large data
start_timer
for (i=0; i<maxiter; i++)
  call mydgemm_batched
end_timer
```



Reproducibility and reliability

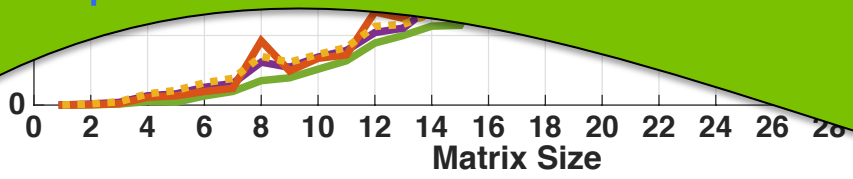



```
initialize (A, B, C);
flush cache large data
start_timer
for (i=0; i<maxiter; i++)
  call mydgemm_batched
end_timer
```



Reproducibility and reliability

The SCALAPACK SVD story
LAPACK: it performs an QR(A) then SVD on $R=U\Sigma V^T$ then $U=Q*U$
Unfortunately Scalapack do not perform this and so comparing tall-skinny SVD against Scalapack is always a win-big.
However a very simple 3 lines of codes can fix the issue
pdgeqrf
pdgesvd
pdormqr



initialize (A, B, C);

A SVD

DRAM

Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results

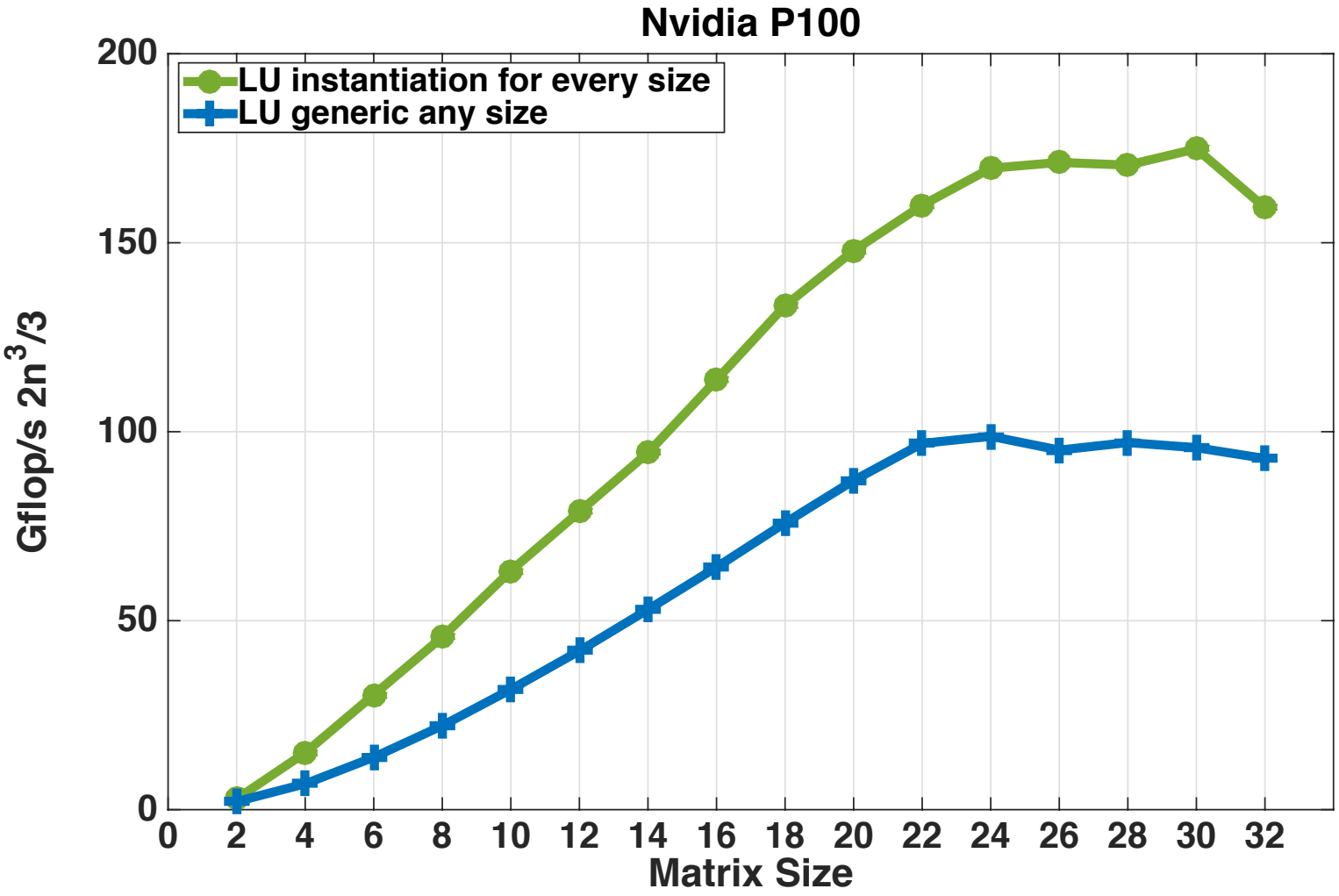
Design: library v.s. paper

- Library need to follow standard interface
  - Which might add many overhead in particular for small matrices
- Library have to be used by developers, applications
- Library have to be generic and accommodate at least most practical cases
- A library (.a .so) cannot be 1 GB, so template instantiation should be limited
- Library have to reliable and robust
- Maybe reproducible accuracy

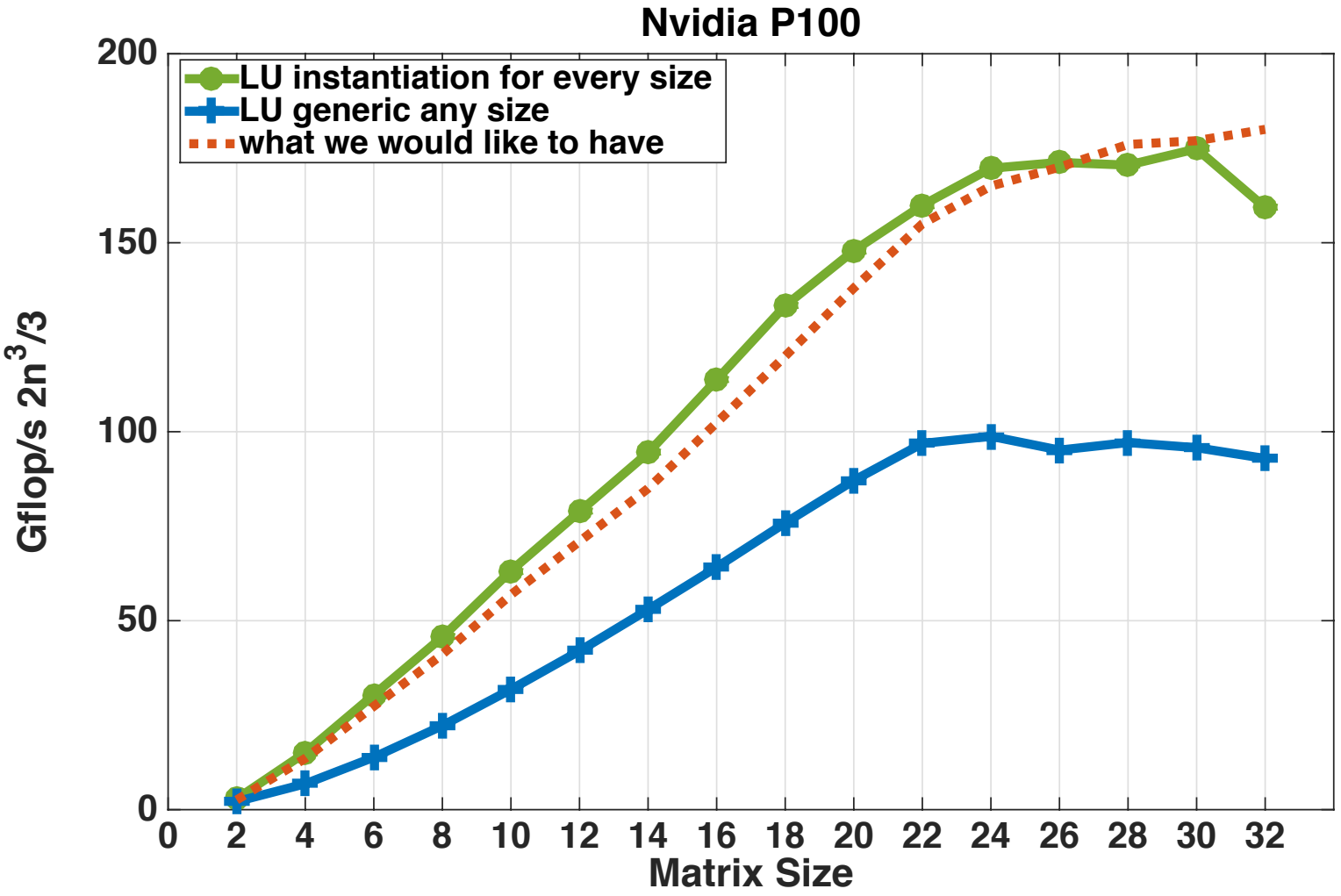
Design: library v.s. paper

- A library should check argument error
- A Library should also check for numerical error and need to be conform with the standard (e.g., Cholesky, LU, and QR need to check for: diag error, singularity, overflow, underflow etc..)
- Papers, posters, proposals:
  - I have seen code without any checking,
  - Even sometimes without accuracy verification, or with self made error checking
  - Most of the time compiled for every size for every run,
  - Work only on a very particular case
  - That's fine when dealing with particular application but cannot be adopted in a library

Design: library v.s. paper



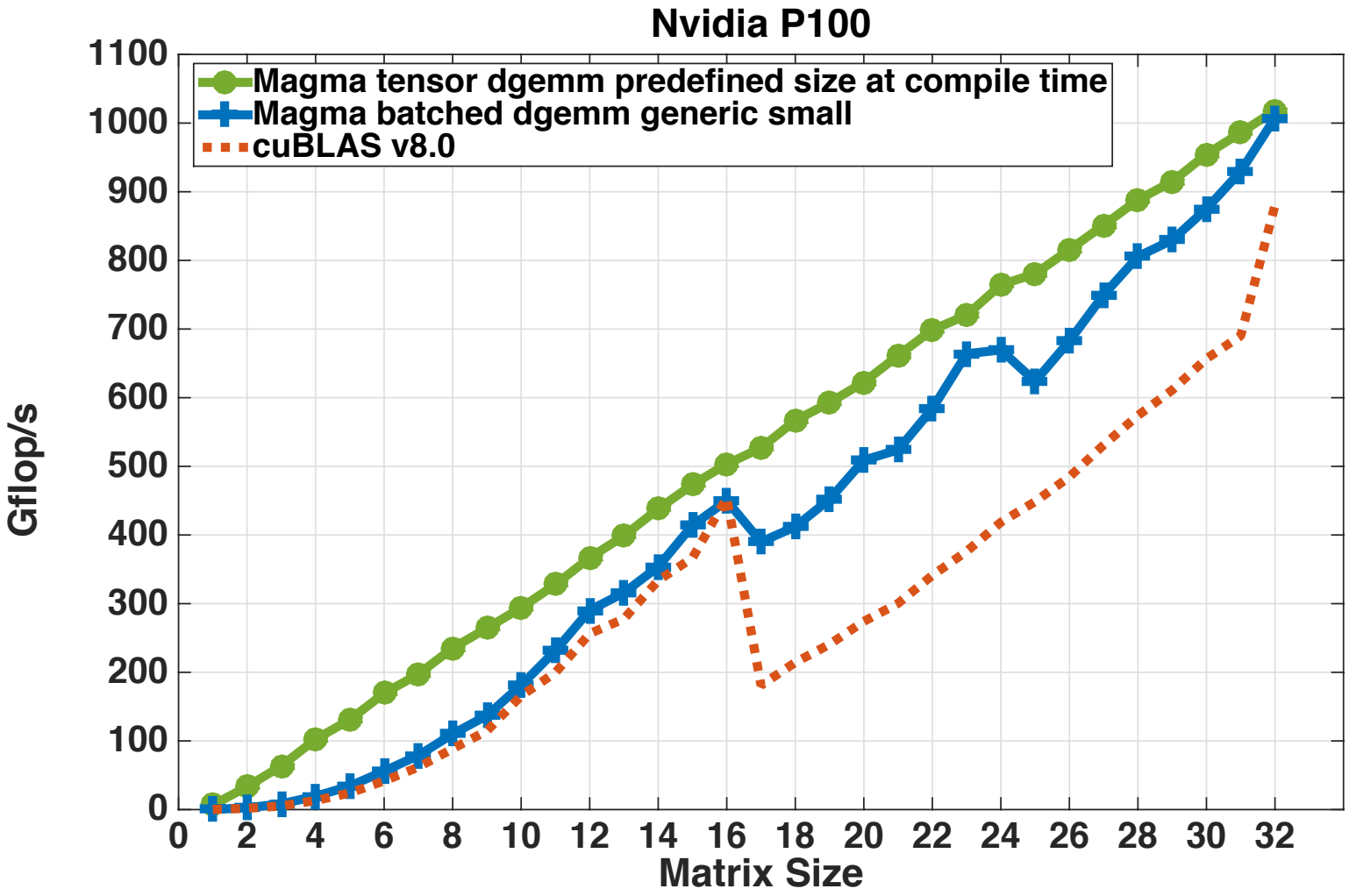
Design: library v.s. paper



ICL

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Design: library v.s. paper



Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results

Design: standard v.s. interleaved

- Interleaved is good but:
  - gemm might work
  - Cholesky, is easy to implement but when the matrix is larger than 16/32 the matrices might not fit into the reg/sm of the SMX, and thus the matrix is going to be reloaded at every update.
  - LU what is going to happen when every matrix has different pivot
  - How to handle variable sizes (gemm, lu, QR LA)
  - Iterative solvers working on different set of batched matrices might converge while the other set still iterating.
  - Multifrontal solver where some data is runtime created and fill in occur
- Standard format showed very good performance and efficiency, but sure effort from analyzing, to modeling, to design to tuning is needed to reach this

Some of my thoughts and observations

- Reproducibility and reliability
- Design: library v.s. paper
- Design: standard v.s. interleaved
- Methodology, Performance Model and Performance Counter Analysis
- Small sizes results

Batched Computations

GPU Optimization Summary

- **Hardware concepts**
  - CUDA core
  - Warp
  - Half-warp
  - Register file
  - Shared memory
  - Atomics
  - Shuffles
  - SMX
- **Software concepts**
  - Stream
  - Thread block
  - Kernel
  - Inlining
  - Intrinsics
- **Algorithmic concepts**
  - Blocking
  - Recursive blocking
  - Kernel replacement
  - Out-of-place operations

Batched Computations

Classical strategies design

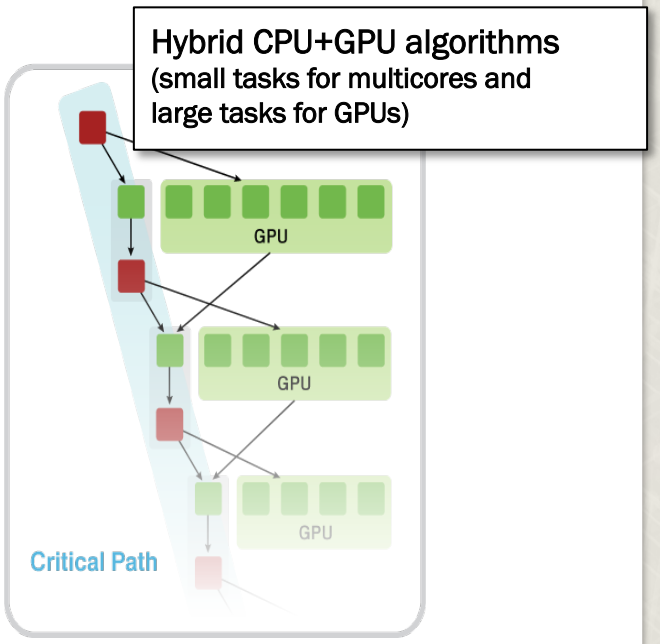
- For large problems the strategy is to prioritize the data-intensive operations to be executed by the accelerator and keep the memory-bound ones for the CPUs since the hierarchical caches are more appropriate to handle it

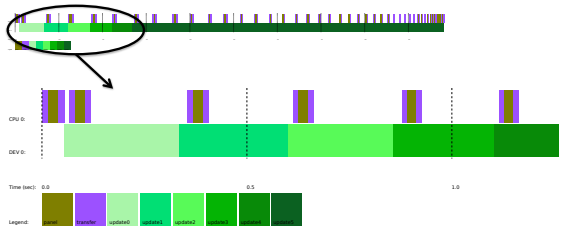
Challenges

- **Cannot be used** here since matrices are very small and communication becomes expensive

Proposition

- **Develop a GPU-only implementation**





Batched Computations

Classical strategies design

- For large problems performance is driven by the Level 3 BLAS (GEMM)

Challenges

- For batched small matrices it is more complicated

Proposition

- **Rethink and Redesign both phases** in a tuned efficient way

Batched Computations

Key observations and current situation:

Classical strategies design

- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory,registers/TB** – load it with data and reuse that data in computations as much as possible.

Challenges

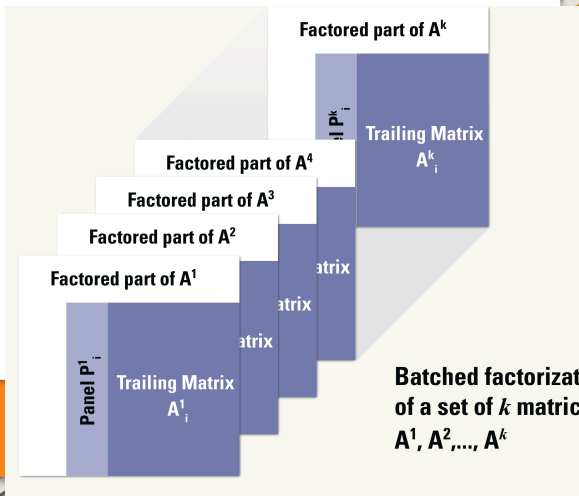
- Our study and experience shows that this procedure provides very good performance for classical GPU kernels but is **not that appealing for batched algorithm** for different reasons.

Batched Computations

Challenges

- Completely **saturating the shared memory** per SMX can decrease the performance of memory bound operations, since only one thread-block will be mapped to that SMX at a time (**low occupancy**)
- Due to the **limited parallelism** in the small matrices, the number of threads used in the thread block will be limited, resulting in low occupancy, and subsequently poor core utilization
- **Shared memory is small** (48KB/SMX) to fit the whole panel
- The panel involves Non-GPU friendly operations:
  - Vectors column (find the max, scale, norm, reduction)
  - Row interchanges (swap)
  - Small number of vectors (apply)

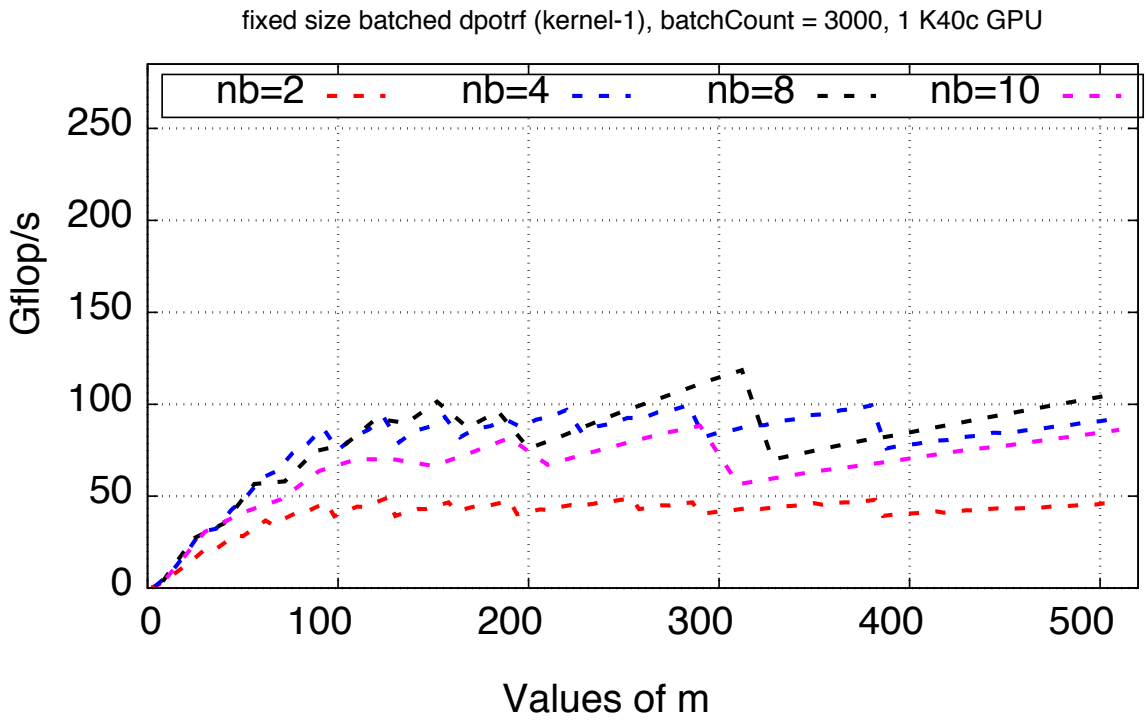
Proposition: custom design per operations type



Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory, registers/TB** – load it with data and reuse that data in computations as much as possible.

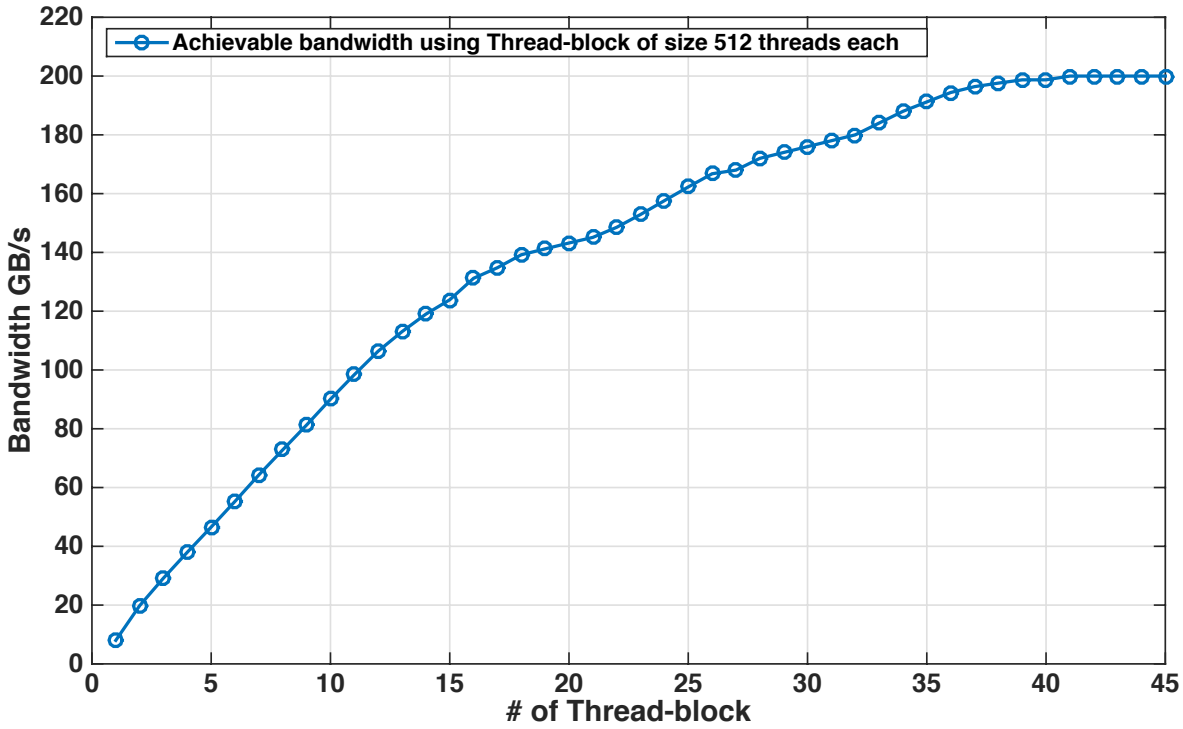


- ✓ **optimized kernel**
- ✓ **using sm/rg**
- ✓ **left v.s. right looking**
- ✓ **autotuned**

Batched Computations

Performance metrics analysis

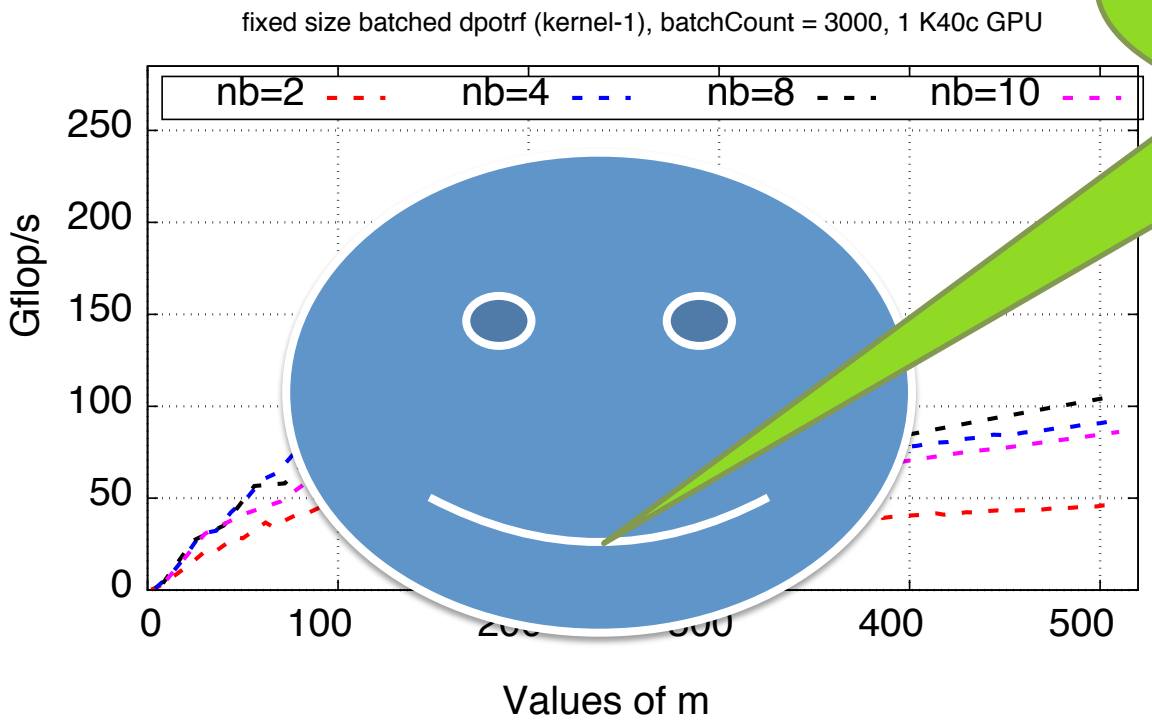
- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory, registers/TB** – load it with data and reuse that data in computations as much as possible.



Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory, registers/TB** – load it with data and reuse that data in computations as much as possible.



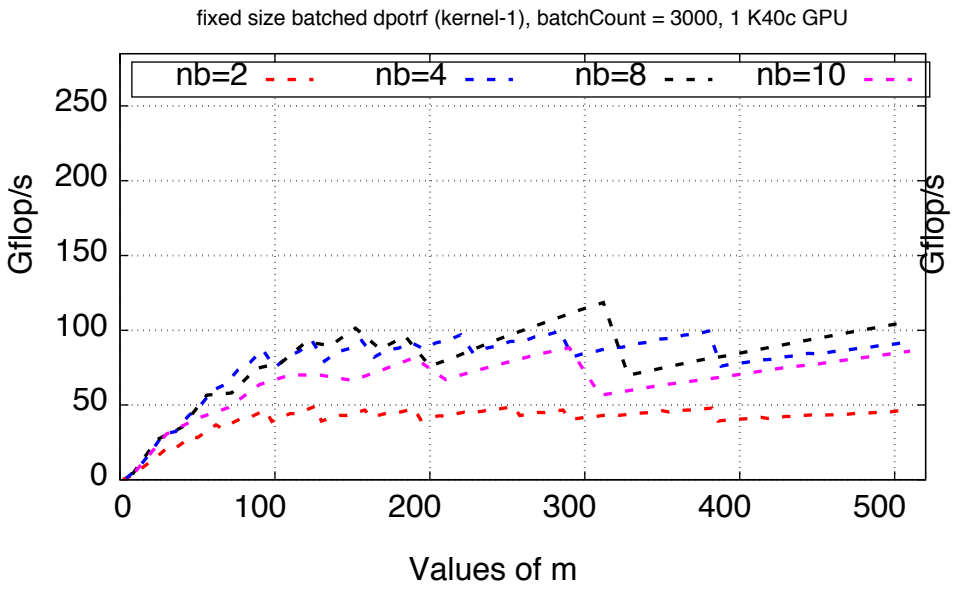
We should focus on the performance analysis and the design of a kernel

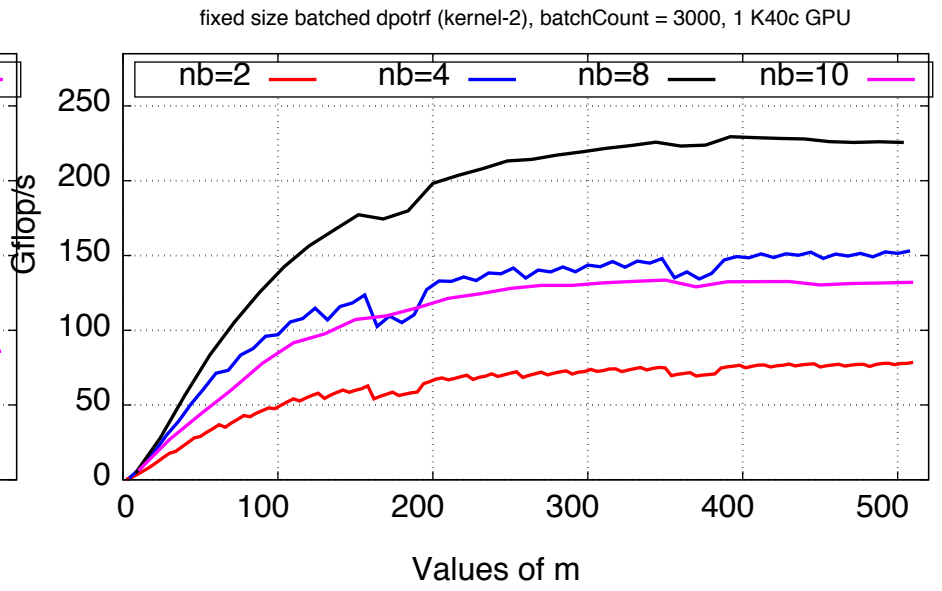
- ✓ **optimized kernel**
- ✓ **using shared memory**
- ✓ **left v.s. right looking**
- ✓ **autotuned**

Batched Computations

Performance metrics analysis

- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory, registers/TB** – load it with data and reuse that data in computations as much as possible.





Methodology, Performance Model and Performance Counter Analysis

$$P_{max} = \frac{F}{T_{min}}$$

Flops for the computation

Fastest time to solution

- For square matrices

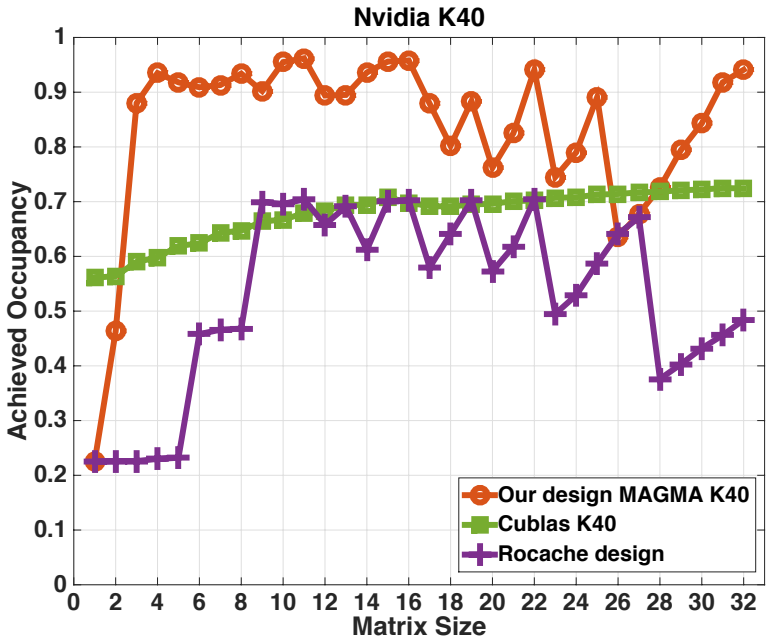
  $F \approx 2n^3$, $T_{min} = min_T\,(T_{Read(A,B,C)} + T_{Compute(C)} + T_{Write(C)})$

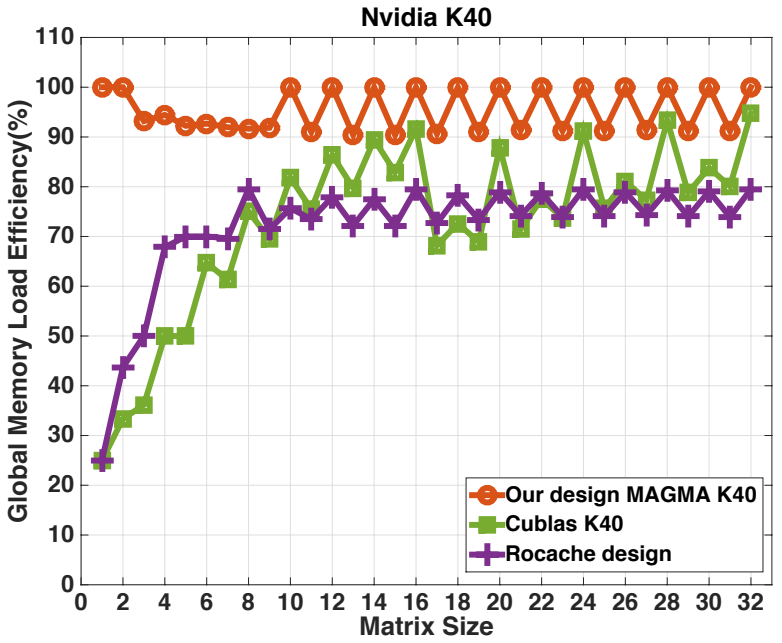
- Need to read/write $4\,n^2$ elements, i.e., $32n^2$ Bytes in DP
  => if max bandwidth is **B**, we can take $T_{min} = 32\,n^2 / B$ in DP. Thus,

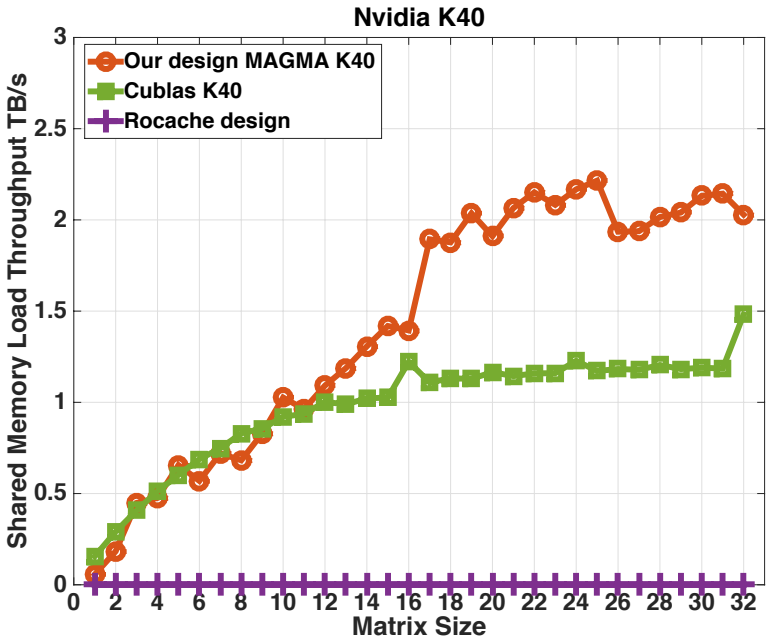
$$P_{max} = \frac{2n^3 B}{32n^2} = \frac{nB}{16} \text{ in DP.}$$

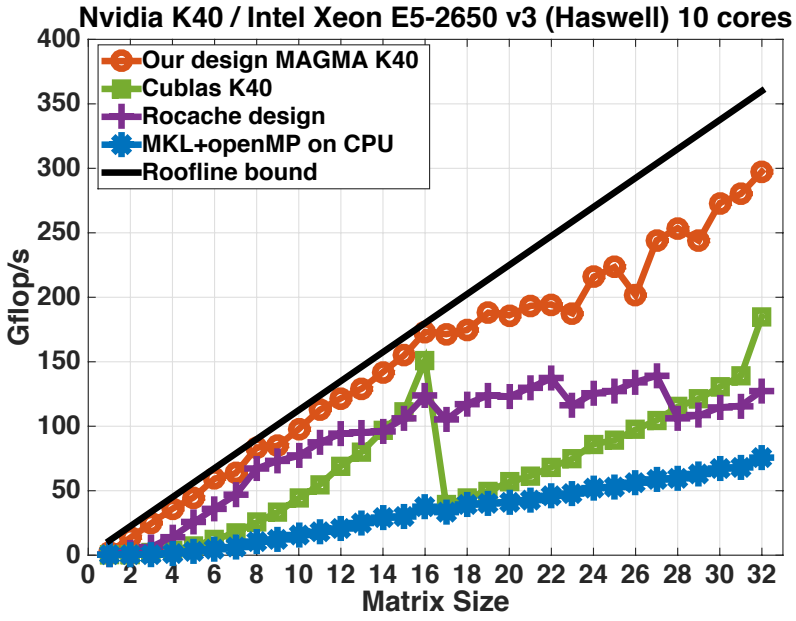
- With ECC on, peak on B on a K40c is ≈180 GB/s, so when n=16 for example, we expect theoretical max performance of 180 Gflop/s in DP

Methodology, Performance Model and Performance Counter Analysis

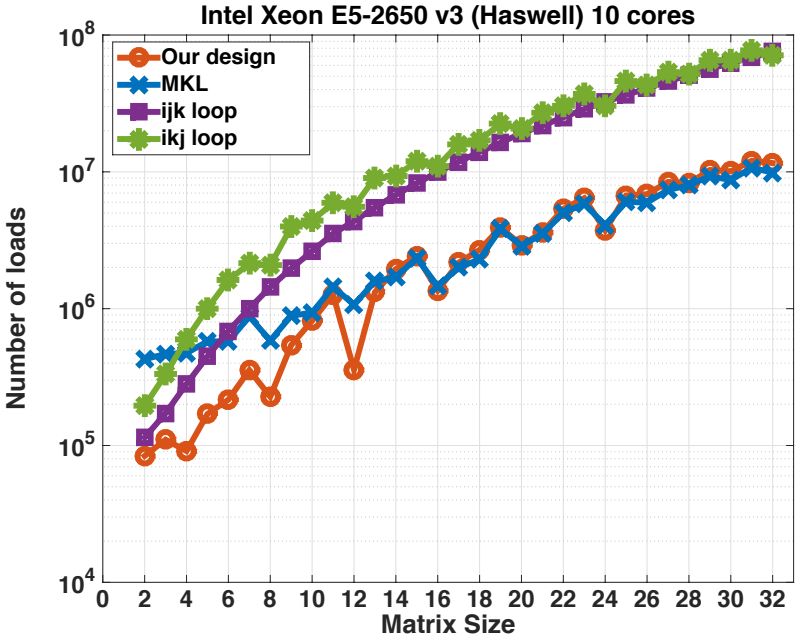


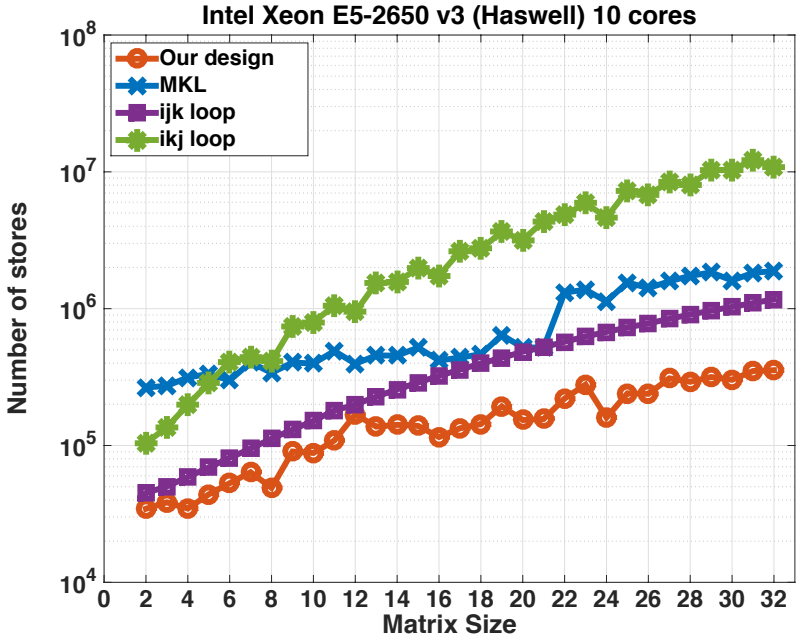


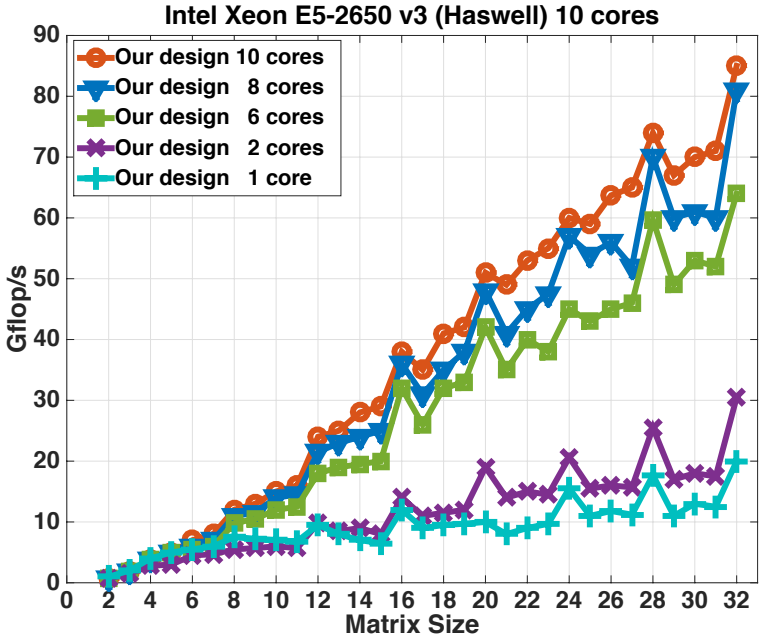


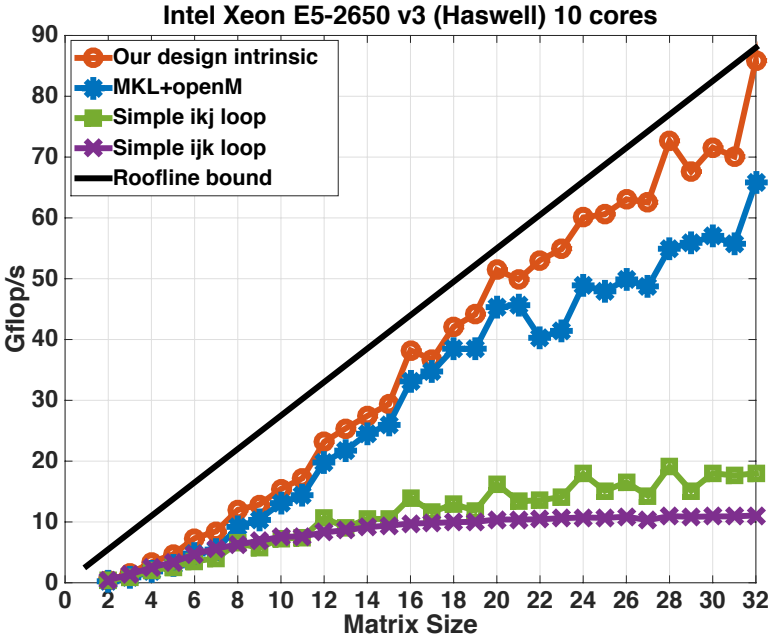


Methodology, Performance Model and Performance Counter Analysis



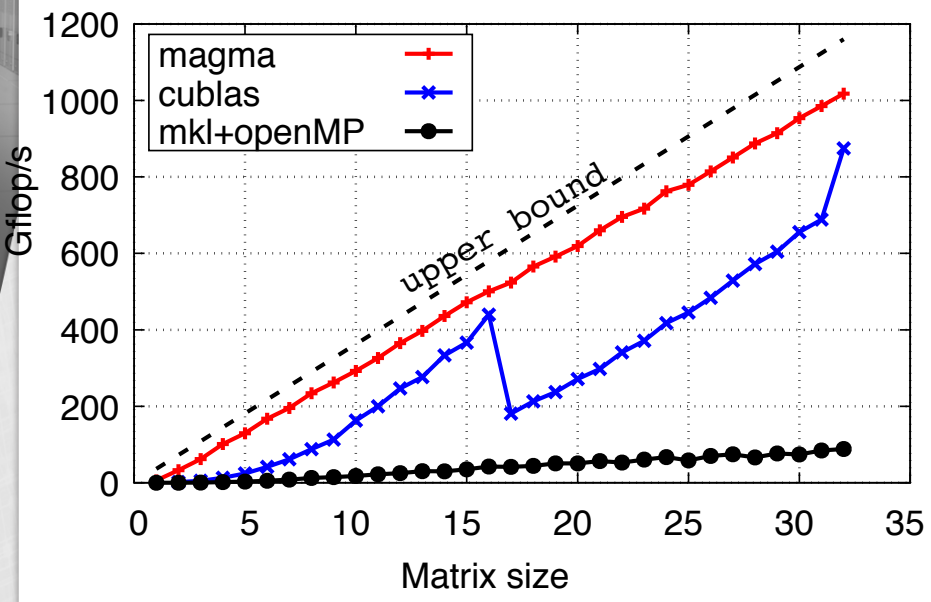




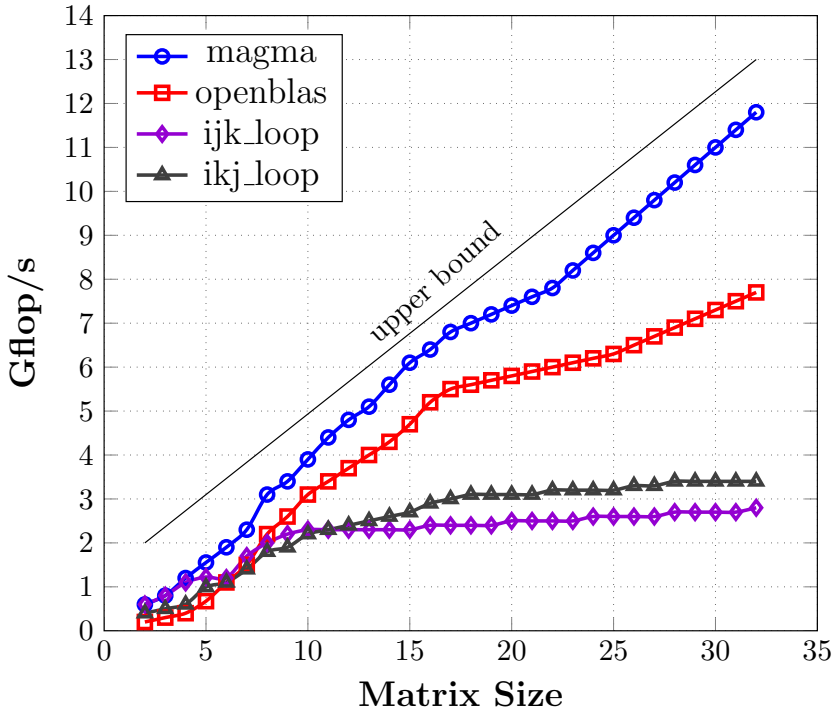


Methodology, Performance Model and Performance Counter Analysis

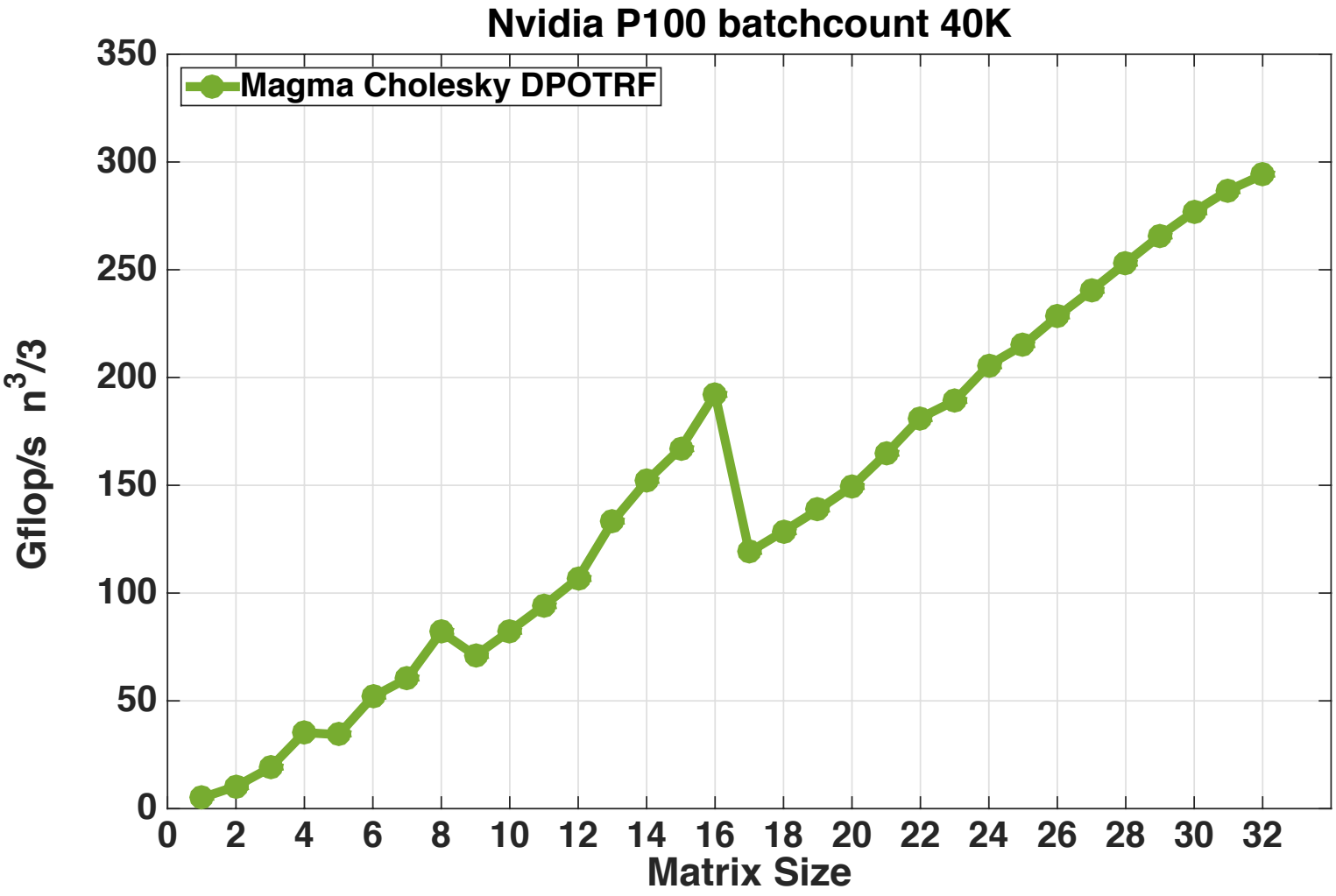
Nvidia P100



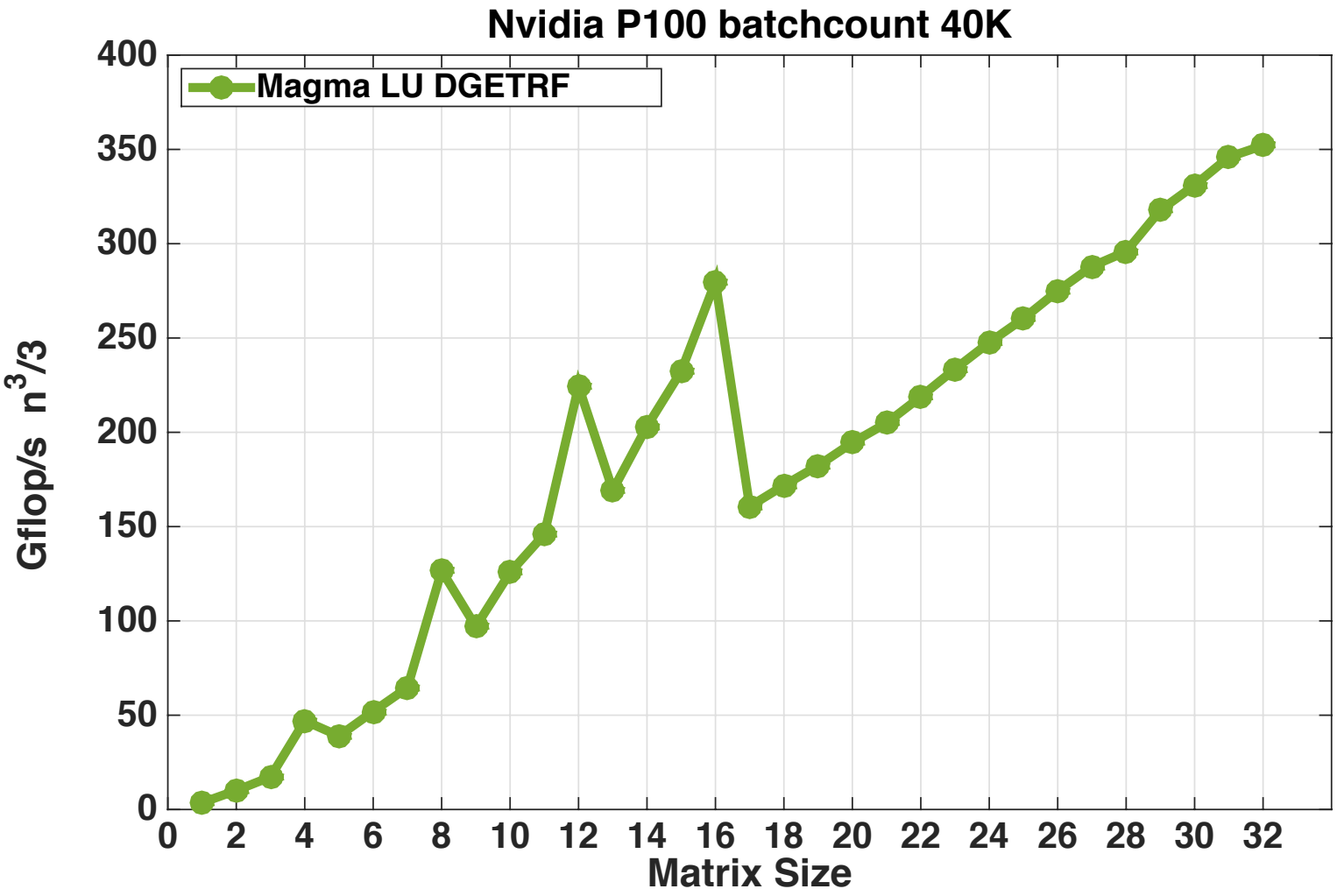
Tegra ARM



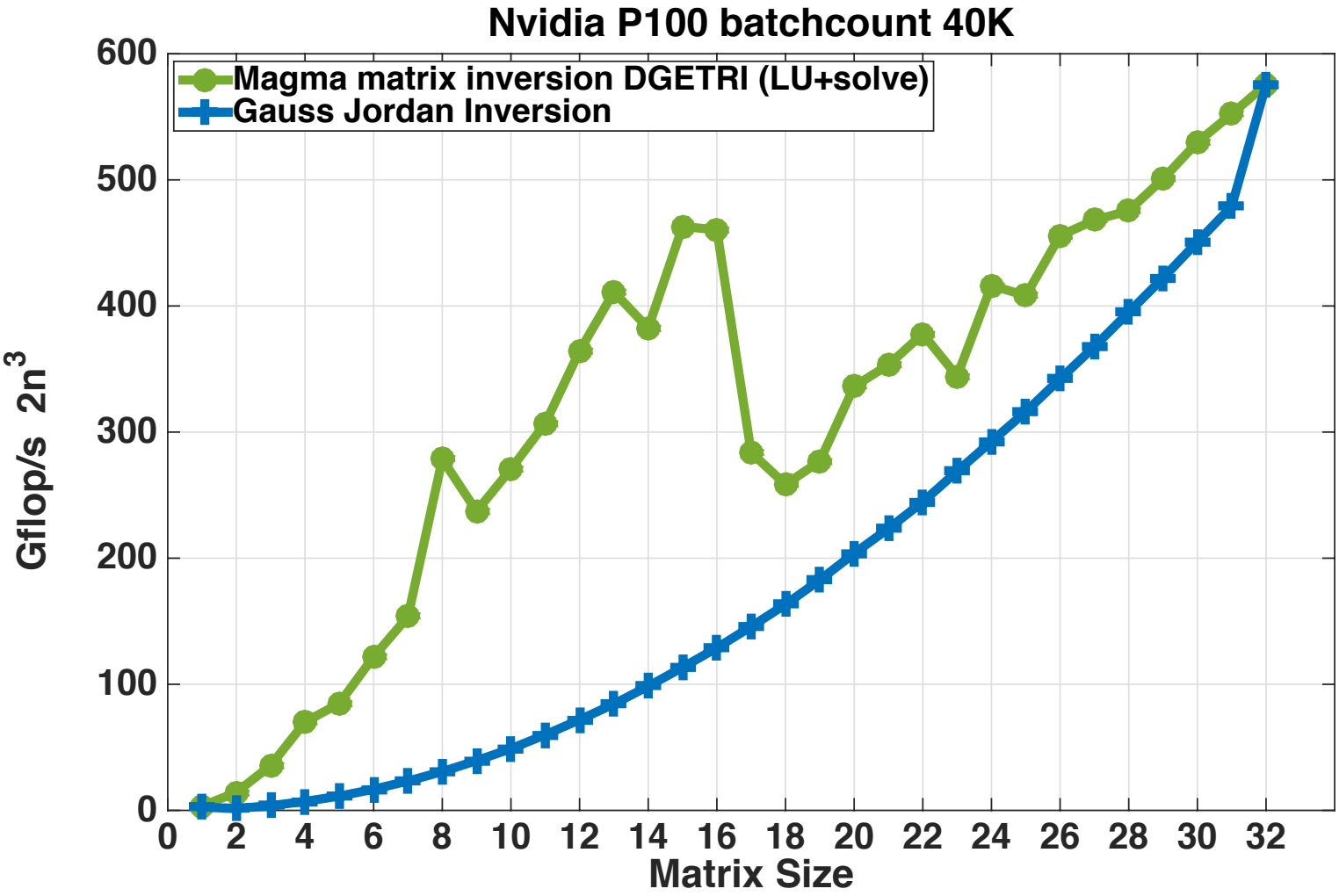
Methodology, Performance Model and Performance Counter Analysis



Methodology, Performance Model and Performance Counter Analysis



Methodology, Performance Model and Performance Counter Analysis



Future trending direction

- Extended functionality and variable sizes
- Customized batched routines for Deep Learning
- FP16 batched routines coming soon
- Sparse components SpDMM, SpMM, SpMV, etc
- Introducing interleaved format
- More Applications specific design
- MAGMA Embedded
- MAGMA DL
- I would encourage a framework for accuracy and performance benchmarking

Collaborators and Support

MAGMA team
http://icl.cs.utk.edu/magma

PLASMA team
http://icl.cs.utk.edu/plasma

Collaborating partners
University of Tennessee, Knoxville
Lawrence Livermore National Laboratory, Livermore, CA
University of California, Berkeley
University of Colorado, Denver
INRIA, France (StarPU team)
KAUST, Saudi Arabia

Umeå University
INRIA
Rutherford Appleton Laboratory
University of Manchester