On the Development of Variable Size Batched Computation for Heterogeneous Parallel Architectures

A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra

To be presented at Parallel and Distributed Scientific and Engineering Computing (PDSEC'16) Workshop

Friday, April 1st, 2016

Outline

1. Introduction
2. Scope of This Work
3. Design Details
4. Experimental Results
5. Conclusion and Future Work

Outline

1. Introduction
2. Scope of This Work
3. Design Details
4. Experimental Results
5. Conclusion and Future Work

Introduction

- Batched routines usually solve a large number of relatively small independent problems
- Each individual problem does not provide sufficient work for the underlying hardware
  - Especially accelerators
- Batching these problems as one task is usually beneficial
- In the **MAGMA** library notation:
  - Fixed size problems (e.g., `magma_dpotrf_batched`)
    - *Haidar et al., IJHPCA 2015*
  - Variable size problems (e.g., `magma_dpotrf_vbatched`)

What is the motivation?

- Many scientific applications can benefit from batched routines

Astrophysics	batched LU
Tensor contractions	batched GEMMs
Large scale sparse direct solvers	batched LU, QR, and Cholesky
CA Krylov solvers	batched GEMM, GEMV and others
Anomaly detection in hyperspectral images	batched Cholesky

- In general, problems may vary in size

Does the existing software help achieve high performance?

- Yes for CPUs, No for GPUs
- CPUs
  - Relatively small matrices can fit into fast caches
  - One core per matrix at a time
- GPUs
  - Throughput oriented with small caches
  - Existing software is hybrid, and targets large matrices
    - CPU $\leftarrow$ panel factorization; GPU $\leftarrow$ updates
    - Updates usually hide the CPU-GPU communication
  - This is not applicable for small matrices
    - Updates are too small to hide any communication
  - A full GPU solution is required

Outline

1 Introduction

2 Scope of This Work

3 Design Details

4 Experimental Results

5 Conclusion and Future Work

Scope of This Work

- A fully GPU-based Cholesky factorization on batches of matrices with different sizes

$$\text{LAPACK} \leftarrow \text{BLAS}$$
$$\text{Batched LAPACK} \leftarrow \text{batched BLAS}$$

- Consequently, we developed vbatched routines for:
  - GEMM: Matrix multiplication
  - SYRK/HERK: Symmetric/Hermitian rank-k updates
  - TRSM: Triangular solve
  - POTF2: Cholesky panel factorization
- Some of these routines are useful for other algorithms

Outline

1 Introduction

2 Scope of This Work

3 Design Details

4 Experimental Results

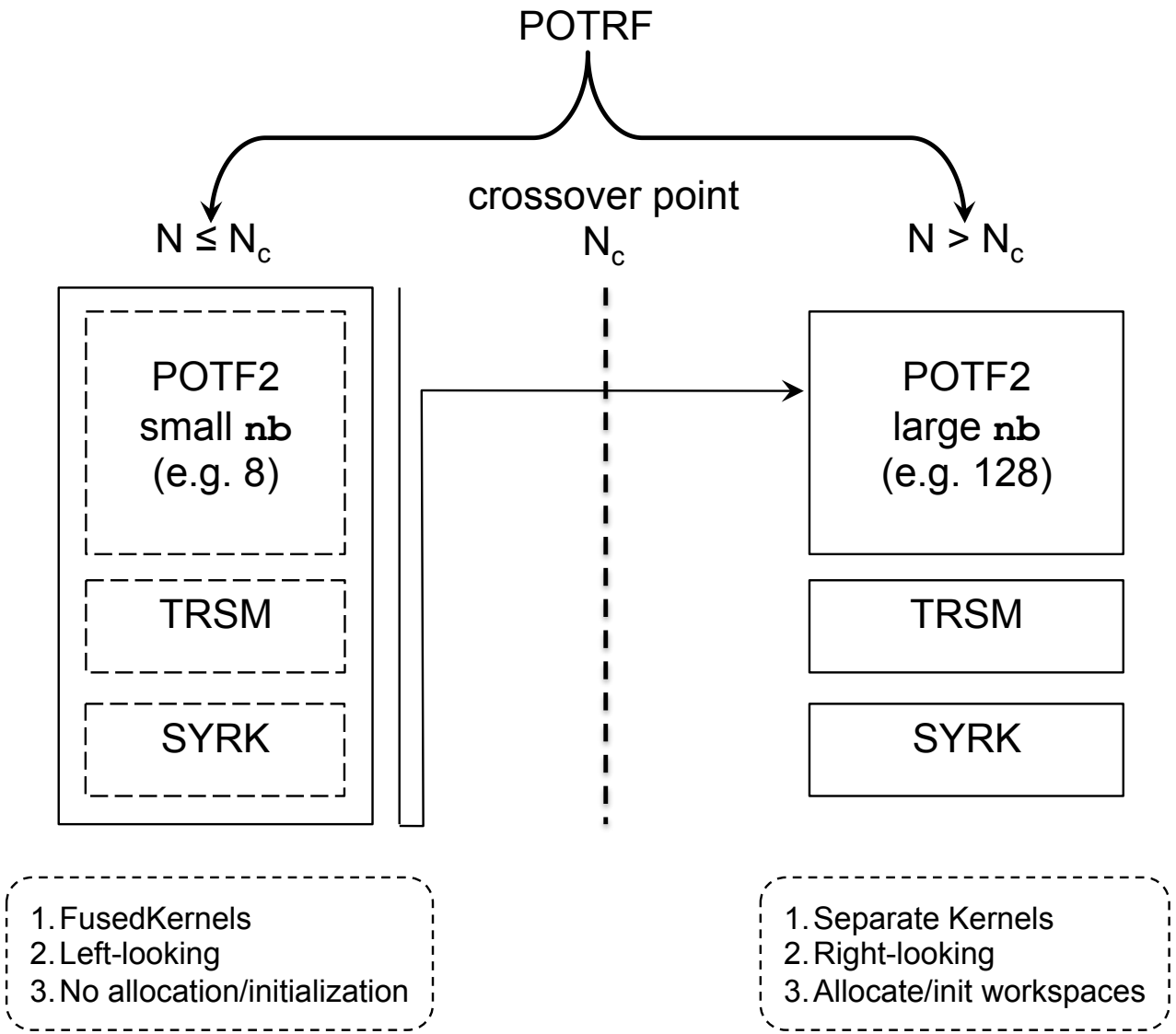
5 Conclusion and Future Work

Proposed Interface

```
magma_dpotrf_vbatched( enum      uplo       ,
                       int       *N         , int Nmax,
                       double    **A_array  , int *ldda,
                       int       *info_array, int batchCount)
```

- All arrays are in the GPU memory
- Each matrix has its own size and leading dimension
- The kernel must accomodate the largest matrix
  - This is why Nmax is needed
  - Nmax can be computed internally with a slight overhead
- Arithmetic operations on sizes are promoted into lightweight GPU kernels

Overall Design

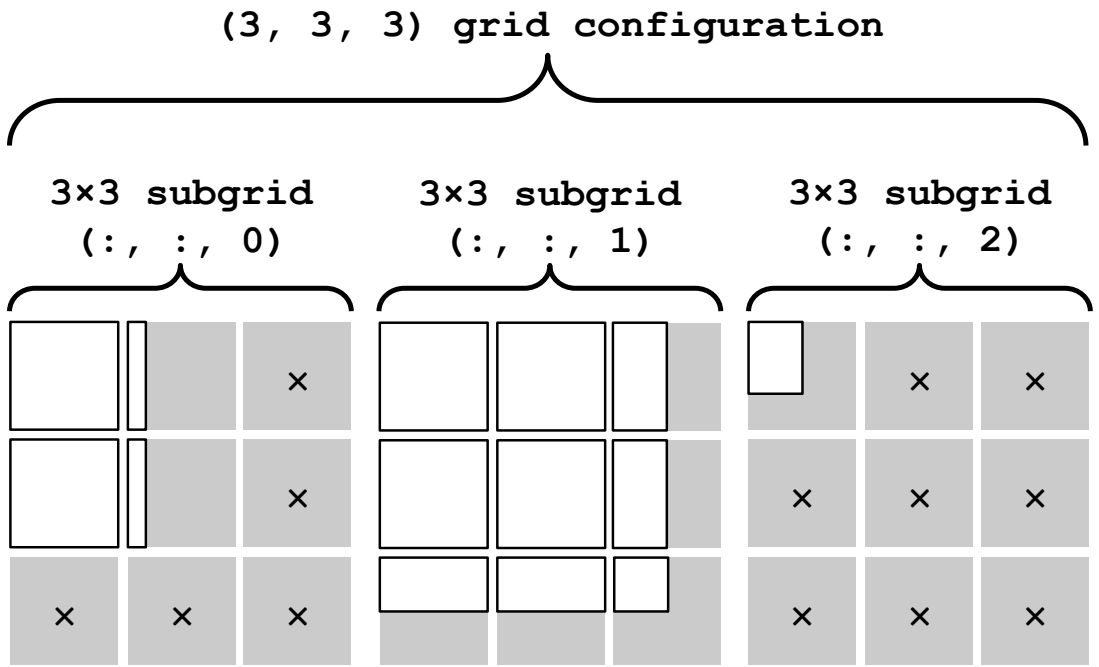


Separate Batched BLAS Approach

- Mainly dominated by SYRK/HERK and TRSM
- Both are dependent on matrix multiplication (GEMM)
  - TRSM: invert block diagonal and perform GEMM accordingly
  - SYRK: normal GEMM, but skip writing upper/lower triangular part
- The MAGMA batched GEMM has been thoroughly tuned for small sizes
  - *Abdelfattah et al., ISC'16*
  - Early Termination Mechanisms are used to support variable sizes
- Sometimes we use cuBLAS SYRK/HERK kernel submitted to concurrent streams

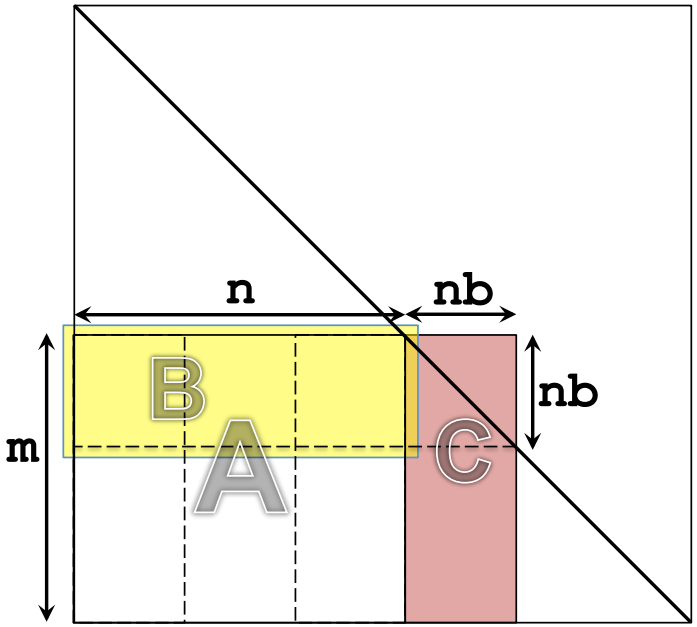
Early Termination Mechanisms (ETMs)

- Kernels are launched to accomodate the largest matrix
- ETMs terminate thread-blocks (TBs) that may not do work for smaller matrices
- Used in GEMM, and consequently, TRSM and SYRK



Fused Batched BLAS Approach

- Used in panel factorization (POTF2)
- Based on left-looking Cholesky factorization
- Fusion of kernels allows data reuse on shared memory
- Other design ideas:
  - Recursive panel approach
  - One factorization step per launch
  - SYRK updates use double buffers



$$C = C - A \times B^T$$

Fused Batched BLAS Approach "cont."

POTF2 for variable sizes

- ETMs
  - Classic: terminates only full TBs
  - Aggressive: also terminales idle threads in live thread blocks
- Implicit sorting
  - Don’t start on all matrices at once
  - At each step, consider problems with sizes within a window of `nb`
- We end up with four versions that share a common code base
  1. ETM classic
  2. ETM classic + implicit sorting
  3. ETM aggressive
  4. ETM aggressive + implicit sorting

Outline

1 Introduction

2 Scope of This Work

3 Design Details

4 Experimental Results

5 Conclusion and Future Work

System Setup

Hardware

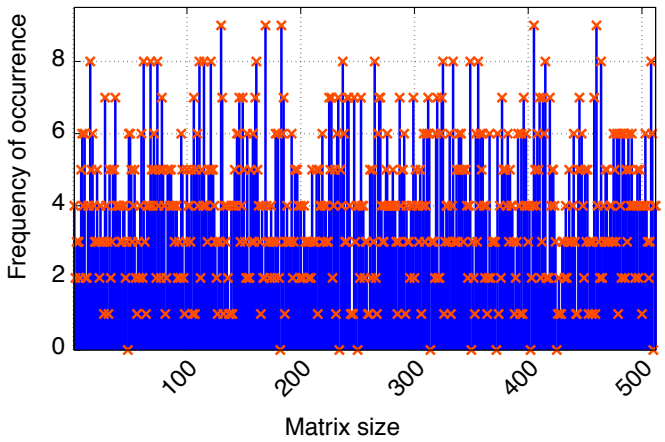
- 2 $\times$ 8-core Sandy Bridge CPUs (Intel Xeon E5-2670 @ 2.6 GHz)
- 1 $\times$ Kepler generation GPU (Tesla K40c @ 745 MHz, ECC on)

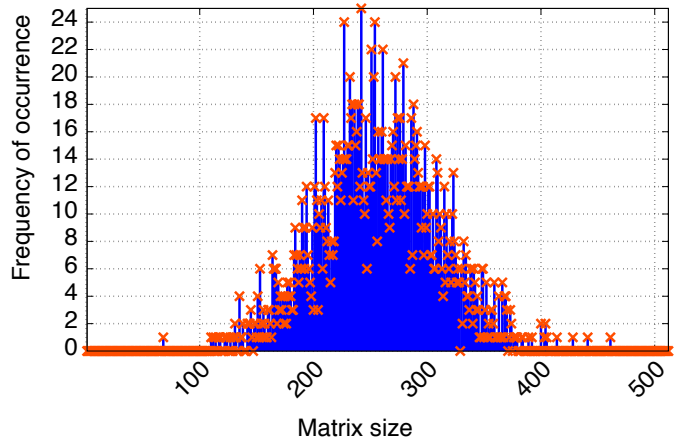
Software

- Intel MKL Library 11.3.0
- CUDA Toolkit 7.0

Matrix Test Cases

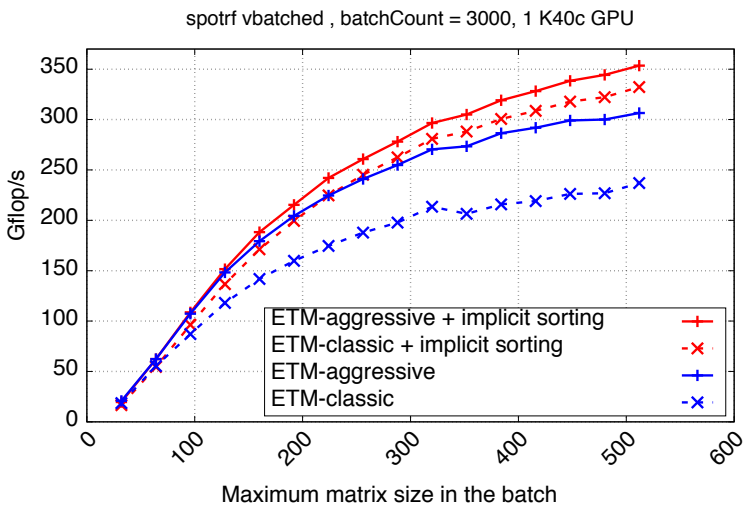
- Randomly generated sizes with uniform/Gaussian distributions

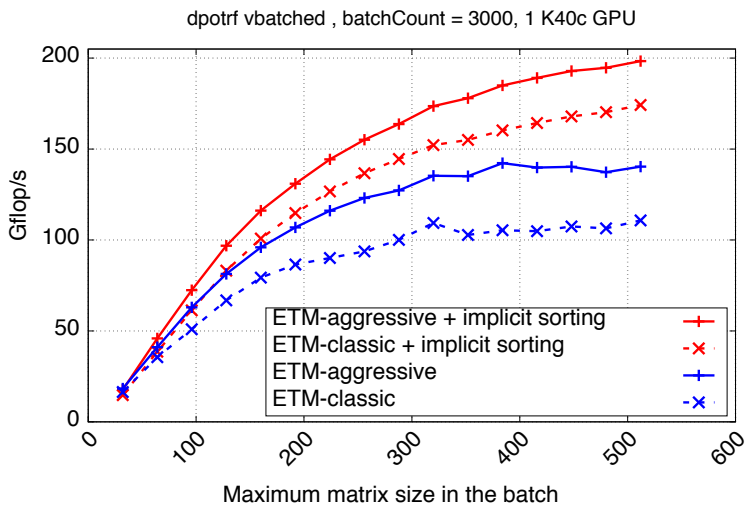




POTF2 Performance

Uniform Distribution

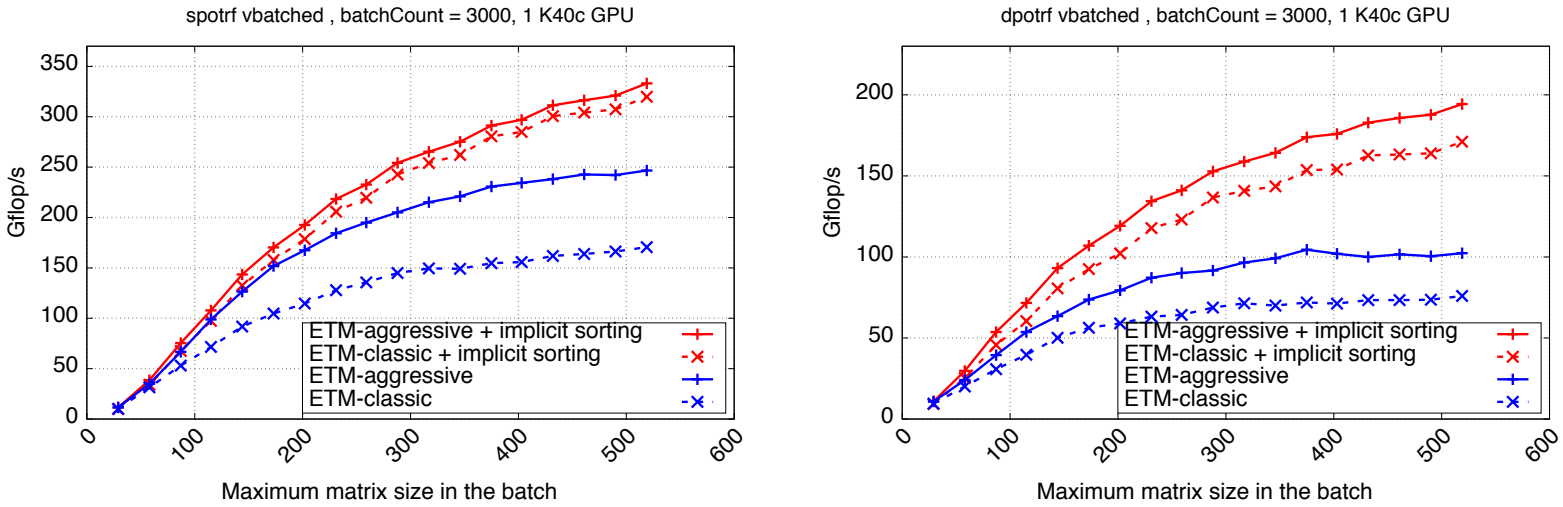




- ETM-aggressive is up to 35% faster than ETM-classic (no implicit sorting)
- Implicit sorting is up to 2× faster

POTF2 Performance "cont."

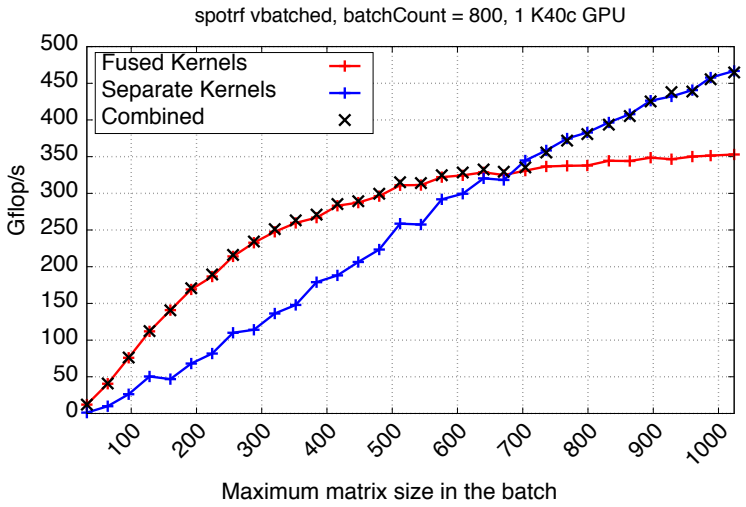
Gaussian Distribution

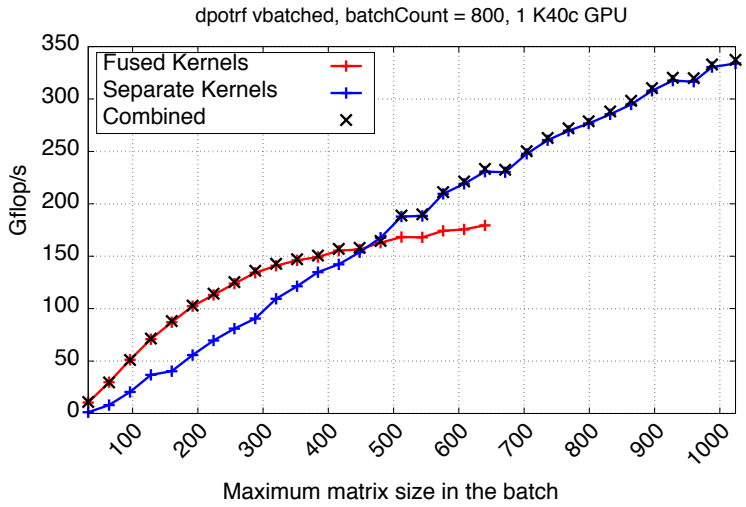


- ETM-aggressive is up to 35% faster than ETM-classic (no implicit sorting)
- Implicit sorting is up to 2× faster

Fused vs. Separate Batched BLAS Approaches

Crossover points

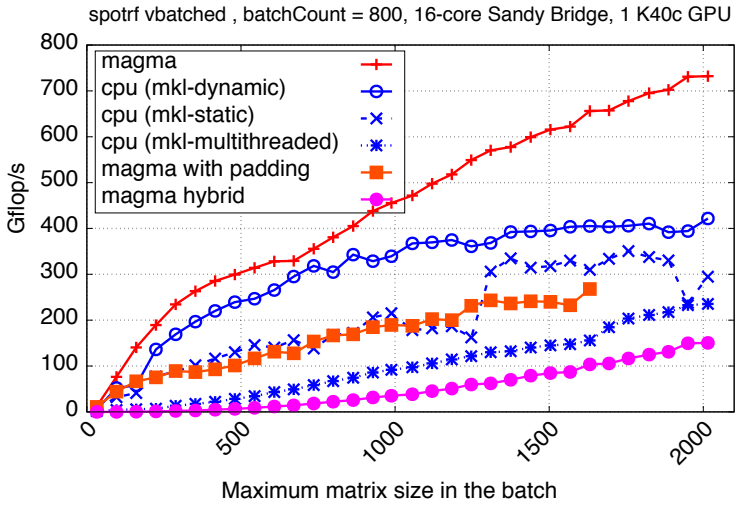


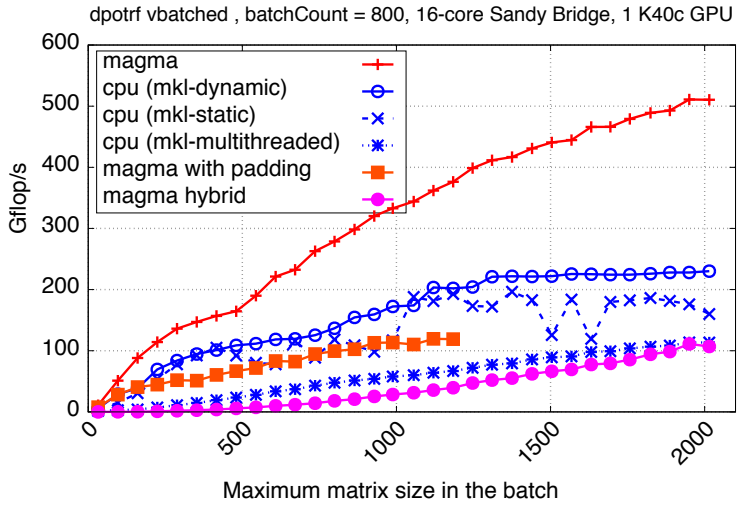


- Fused batched BLAS is up to 2× for small sizes

Overall Performance

Uniform Distribution

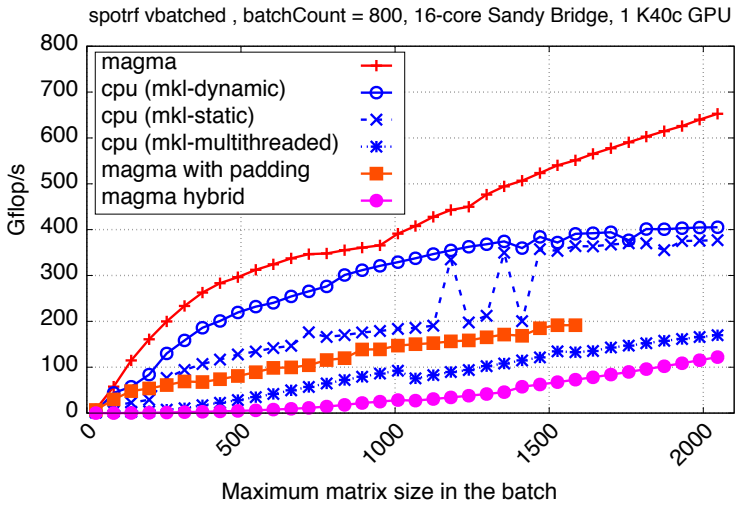


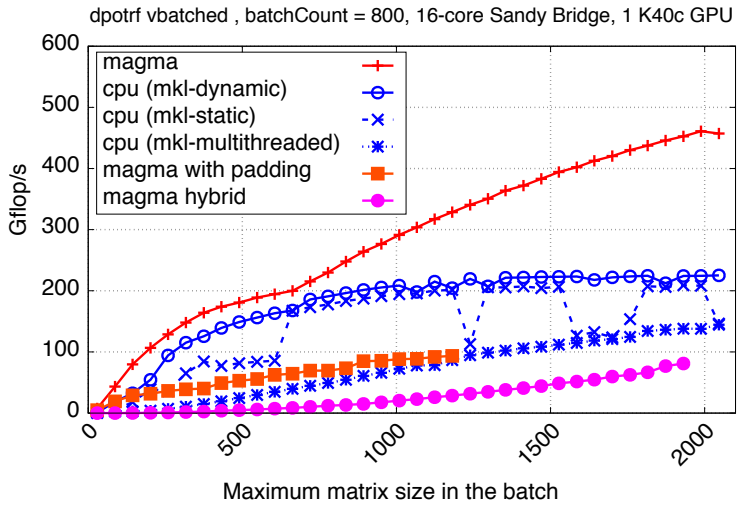


- Best competitor is single-threaded MKL + dynamically unrolled OpenMP loop
- MAGMA-vbatched is generally 2× fatser

Overall Performance

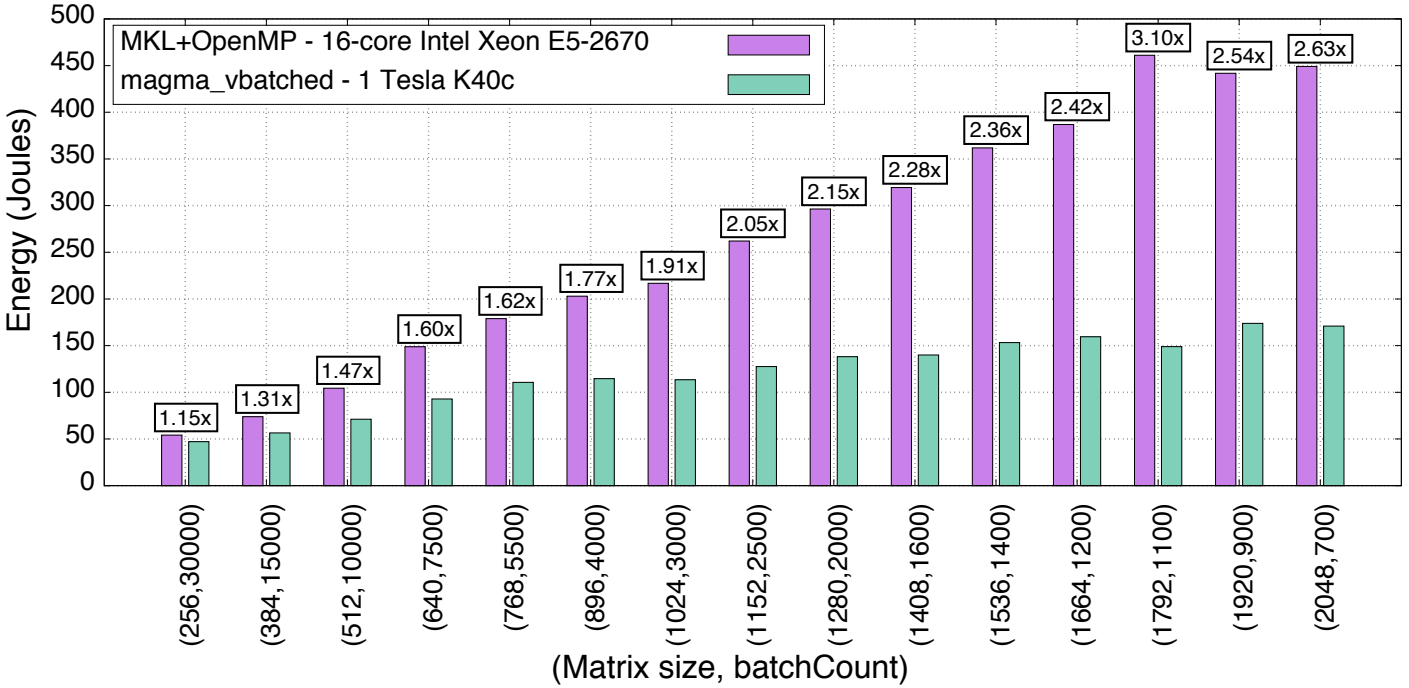
Gaussian Distribution





- Best competitor is single-threaded MKL + dynamically unrolled OpenMP loop
- MAGMA-vbatched is generally 2× fatser

Energy Efficiency



- Using PAPI and NVML
  - Integration of power measurements (sampled every 100ms)
- For every size range, we increase `batchCount` as much as possible
- GPU is up to 3× more energy efficient

Outline

1 Introduction

2 Scope of This Work

3 Design Details

4 Experimental Results

5 Conclusion and Future Work

Conclusion and Future Work

To summarize:

- There is a need for new techniques to develop GPU-accelerated batched routines
- GPUs can achieve many-fold speedup against CPUs, while being more energy efficient
- Kernel design and tuning are a must

Future Directions:

- vbatched LU and QR
- Study the impact of many size distributions on performance
- Applications

Thank You!