Towards Variable Size Batch BLAS (and LAPACK)

A sneak peek into MAGMA's *vbatched* routines

Ahmad Abdelfattah, Azzam Haidar, and Stan Tomov

Workshop on Batched, Reproducible, and Reduced Precision BLAS

Wednesday, May 18th, 2016

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Outline

Outline

Motivation

Why do we need *vbatched* computation?

- In general, applications may require solving many independent problems that are different in size
  - Using fixed size approach with padding is not a good idea
- A typical example is sparse direct solvers
  - Ongoing work with Tim Davis (TAMU)
  - QR, LU, and Cholesky on many relatively small matrices of different sizes

Do we have good starting points?

CPUs: YES

- Problems fits in the CPU relatively large caches
- Optimized routines (e.g. from MKL) can be used
- Use one core per matrix at a time

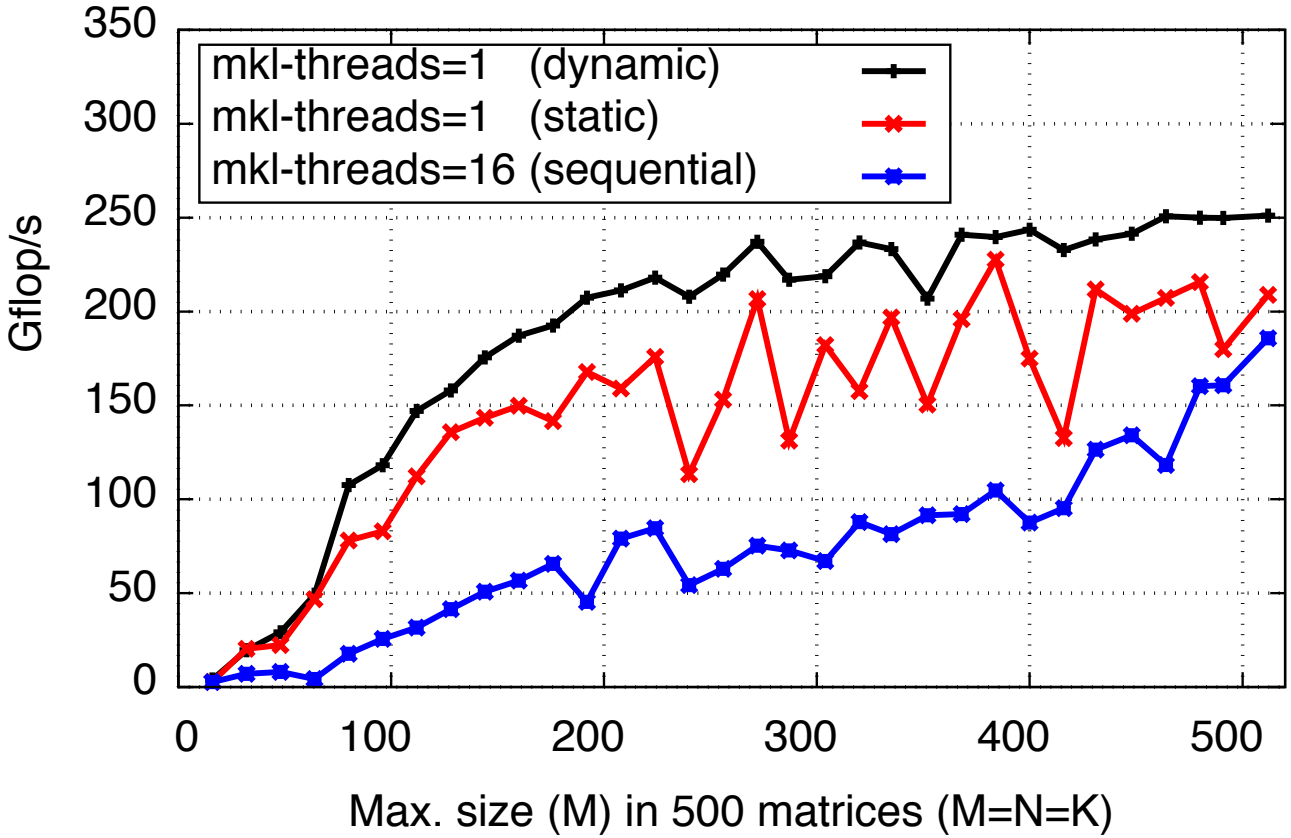
```
// set BLAS #threads = 1
// set OpenMP #threads = 8, 16, 32, ... etc

#pragma omp parallel for schedule(dynamic)
for (int i=0; i < batchCount; i++){
   cpu_dgemm( TRANSA[i], TRANSB[i],
              &M[i], &N[i], &K[i],
              &alpha[i], Aarray[i], &lda[i],
                         Barray[i], &ldb[i],
              &beta[i] , Carray[i], &ldc[i] );
}
```

Do we have good starting points? “cont."

CPUs: YES

Example: DGEMM using MKL + OpenMP (16-core Intel Xeon E5-2690 @ 2.9 GHz)
Batches of square matrices with random sizes



Do we have good starting points? “cont."

GPUs: NO

- No large caches
- Most existing BLAS kernels are designed/tuned for large matrices
  - Use heavyweight thread blocks (TBs)
  - In batched workloads, throughput of TB execution is crucial to performance
- MAGMA uses hybrid (CPU + GPU) approach for most factorization and solve routines
  - Communication kills the performance for small matrices
- A full GPU solution is required
  - We always start by developing the fixed size variant of the routine

Outline

Status of the MAGMA *vbatched* routines

- ***BLAS***
  - Level-3
    - Ready :GEMM, TRSM, SYRK, HERK, SYR2K, and HER2K
    - Dev. :TRMM, SYMM, and HEMM
  - Level-2
    - Ready :GEMV
    - Dev. :SYMV
  - Level-1
    - Ready :AXPY
- ***LAPACK***
  - One-sided Factorization
    - Ready :Cholesky (POTRF)
    - Dev. :LU (GETRF) and QR (GEQRF)

Interface of MAGMA's vbatched routines

- Variable size BUT the same problem settings (for now)
  - e.g., TRANSX, UPLO, and SIDE are all the same
  - Different settings are doable, but might impact performance.

```
void magmablas_dgemm_vbatched(
     enum transA, enum transB,
     int* m, int* n, int* k,
     double alpha, double ** Aarray, int* lda,
                   double ** Barray, int* ldb,
     double beta,  double ** Carray, int* ldc,
     int batchCount, magma_queue_t queue );
```

Outline

Two Design Approaches

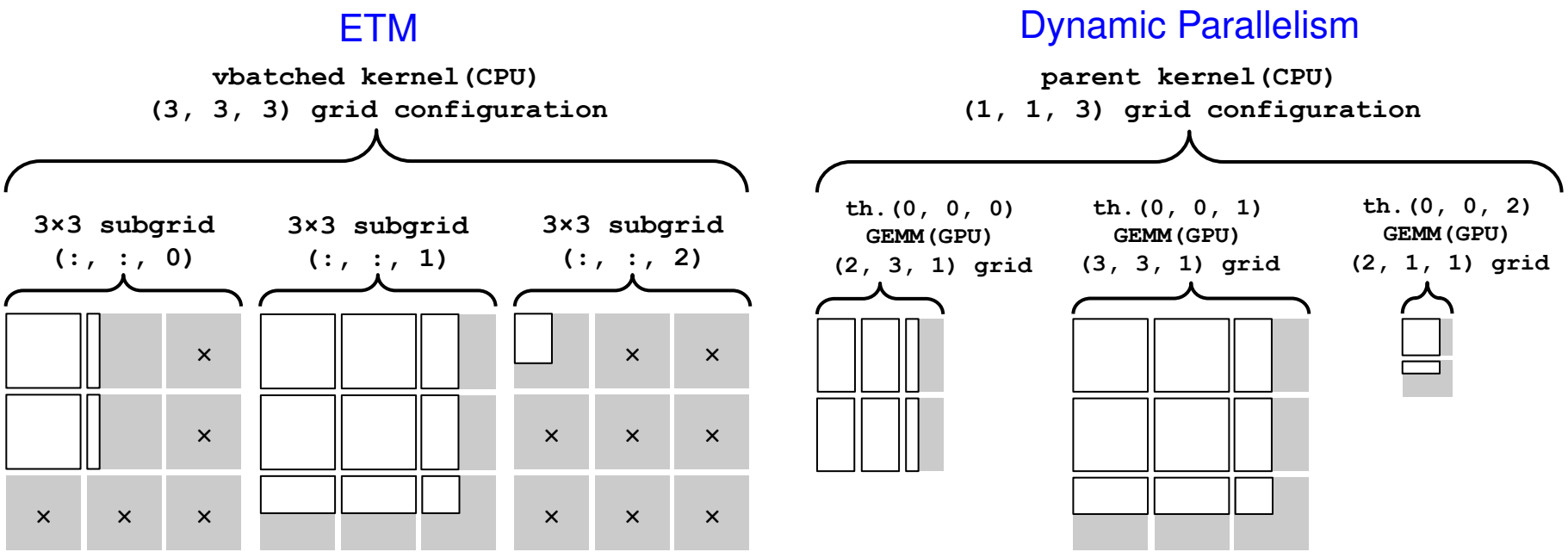
1. Early Termination Mechanisms (ETMs)
   - Computational kernel is launched by the CPU
   - The kernel must accomodate the biggest matrix
     - e.g., we need `max_M`, `max_N`, and `max_K` in `GEMM`
   - ETMs identifies threads with no work to do for smaller matrices
   - Pros
     - Minor overhead (only one kernel launch)
     - The CPU decides the tuning parameters
   - Cons
     - Needs the maximum dimensions
       (exposed to user vs. computed internally)
     - The selection criteria for tuning parameters is tricky
       (right now, based on the maximum dimensions)

Two Design Approaches "cont."

2. Dynamic Parallelism
   - Computational kernels are launched on the GPU
   - The CPU launches one master kernel with as many threads `batchCount`
   - Each master thread launches a kernel for one matrix
   - Pros
     - No need to know the maximum dimensions
     - Choose the best tuning parameters for each matrix
   - Cons
     - Large overhead (launches `batchCount+1`) kernels
     - At least compute capability 3.5

ETM vs. Dynamic Parallelism

Consider a GEMM kernel for with `batchCount = 3`



Optimization Techniques

1. Conventional techniques
   - Register blocking, prefetching, loop unrolling, ... etc
   - BUT not necessarily maximizing resources per TB
2. Kernel Fusion
   - Fused batched BLAS kernels for very small matrices
   - Saves global memory traffic
3. Implicit Sorting
   - Similar computational steps called several times
     (e.g. panel/update)
   - At each step, consider sizes within a relatively small window
   - Delay processing of smaller matrices
     (load balancing)

GEMM Tuning: Aiming for The Best Performance

- Two phases of tuning
  1. Tuning the MAGMA-BLAS kernel
     - Selection of the best tuning parameters according to problem setting/precision/sizes
     - Consider typical test cases found in LAPACK
     - Further details: *Abdelfattah et al., ISC'16*
  2. Selecting the best kernel out there
     - Two MAGMA-BLAS kernels (ETM vs dynamic parallelism)
     - Streamed cuBLAS GEMM

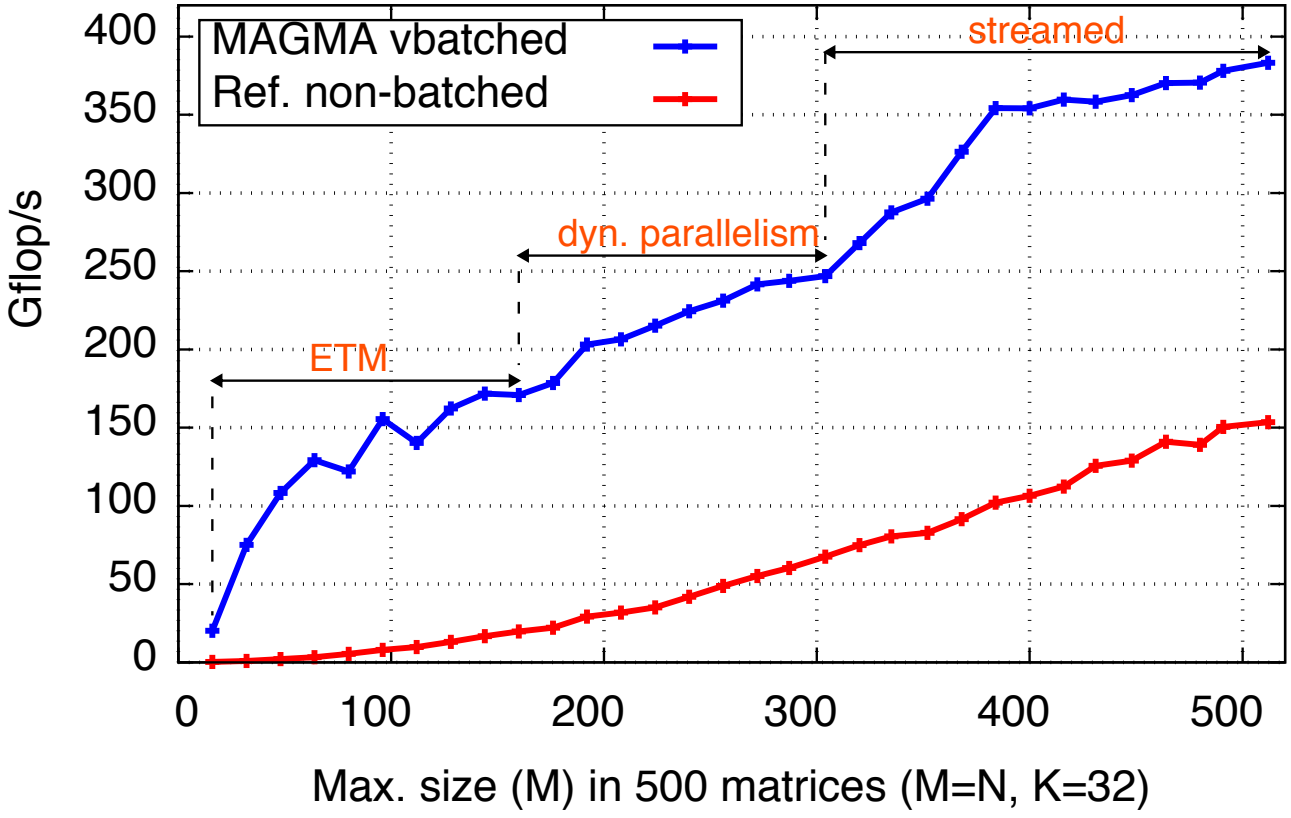
Eventually, we want the best performance when we call `magma_Xgemm_vbatched`

Outline

Experimental Setup

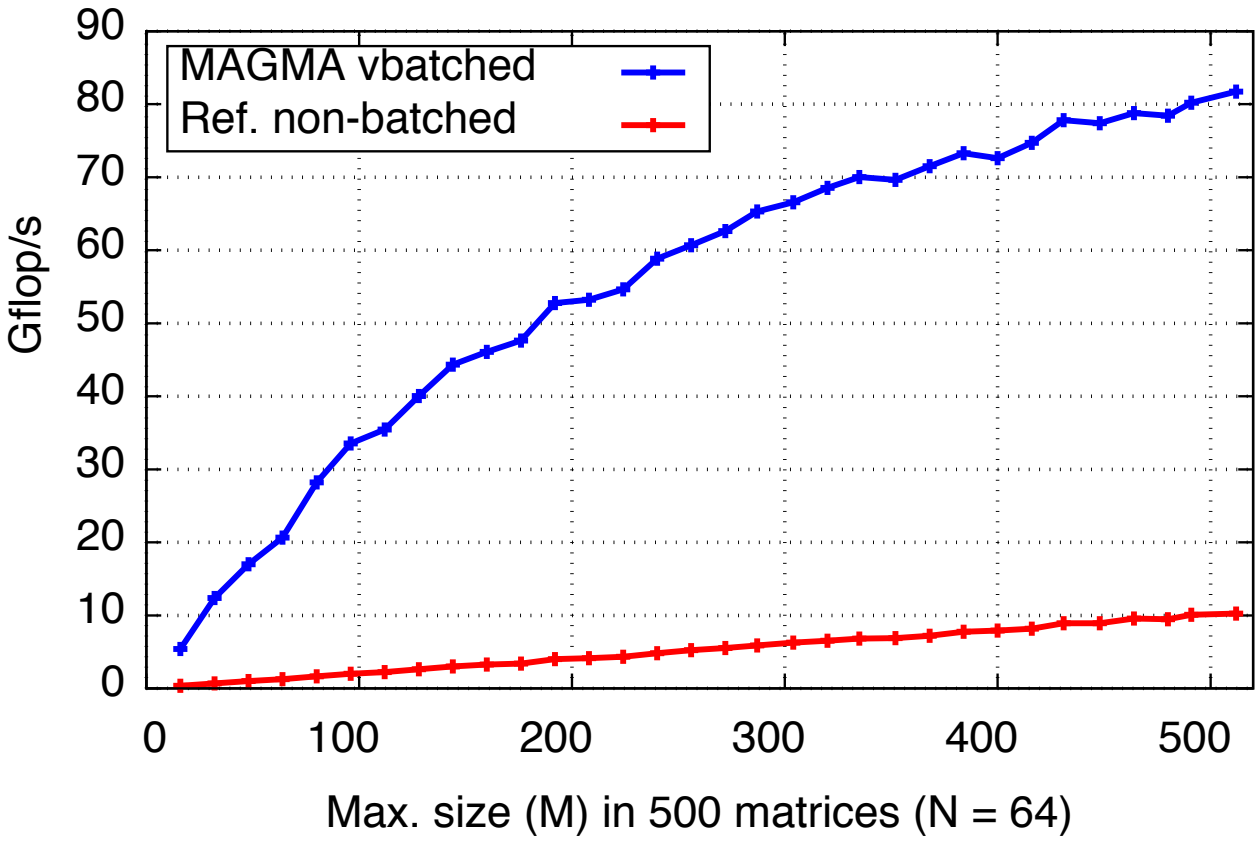
- 1 $\times$ Kepler generation GPU (Tesla K40c @ 745 MHz, ECC on)
- CUDA Toolkit 7.0
- Randomly generated sizes with a uniform distribution
- Performance Comparisons
  - MAGMA-vbatched vs. non-batched reference implementations (using cuBLAS)

Matrix-matrix Multiplication (DGEMM)



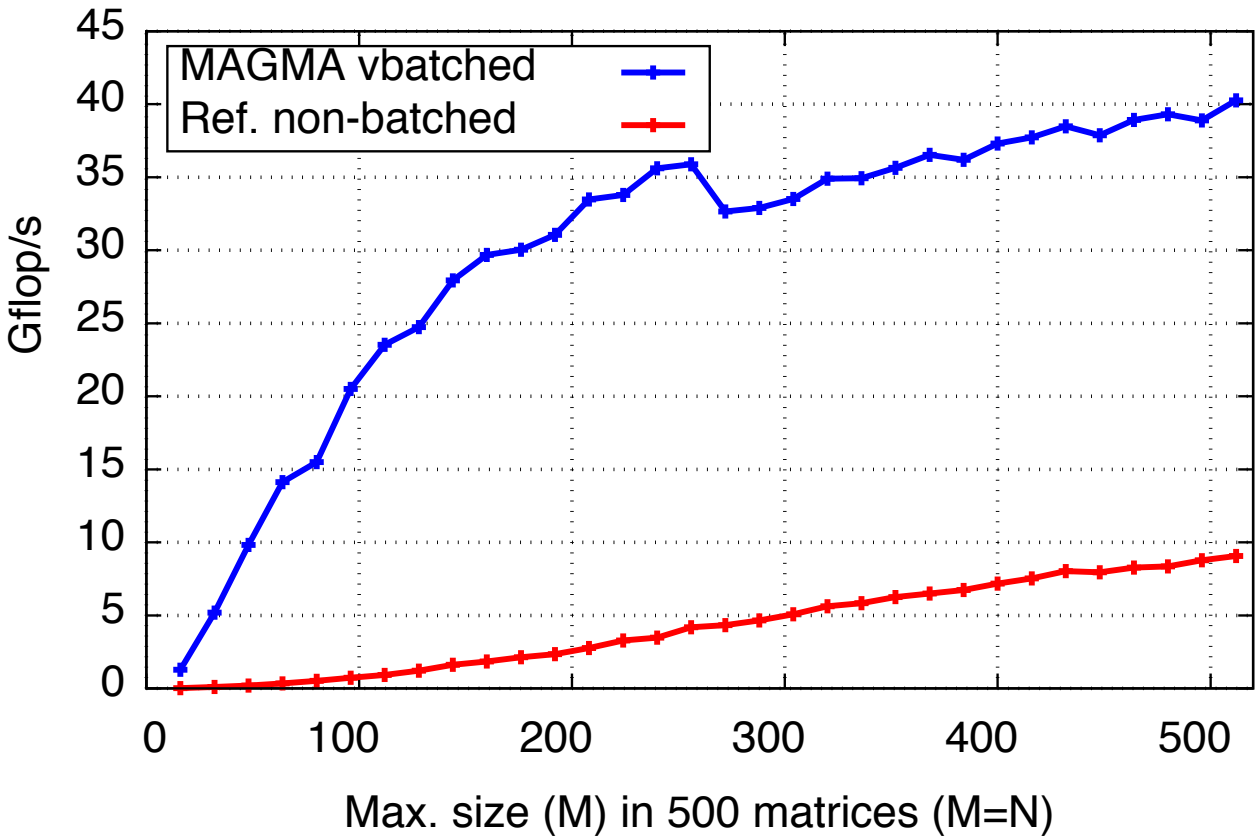
- Speedups from 2.5× up to 50+×

Triangular Solve (DTRSM)



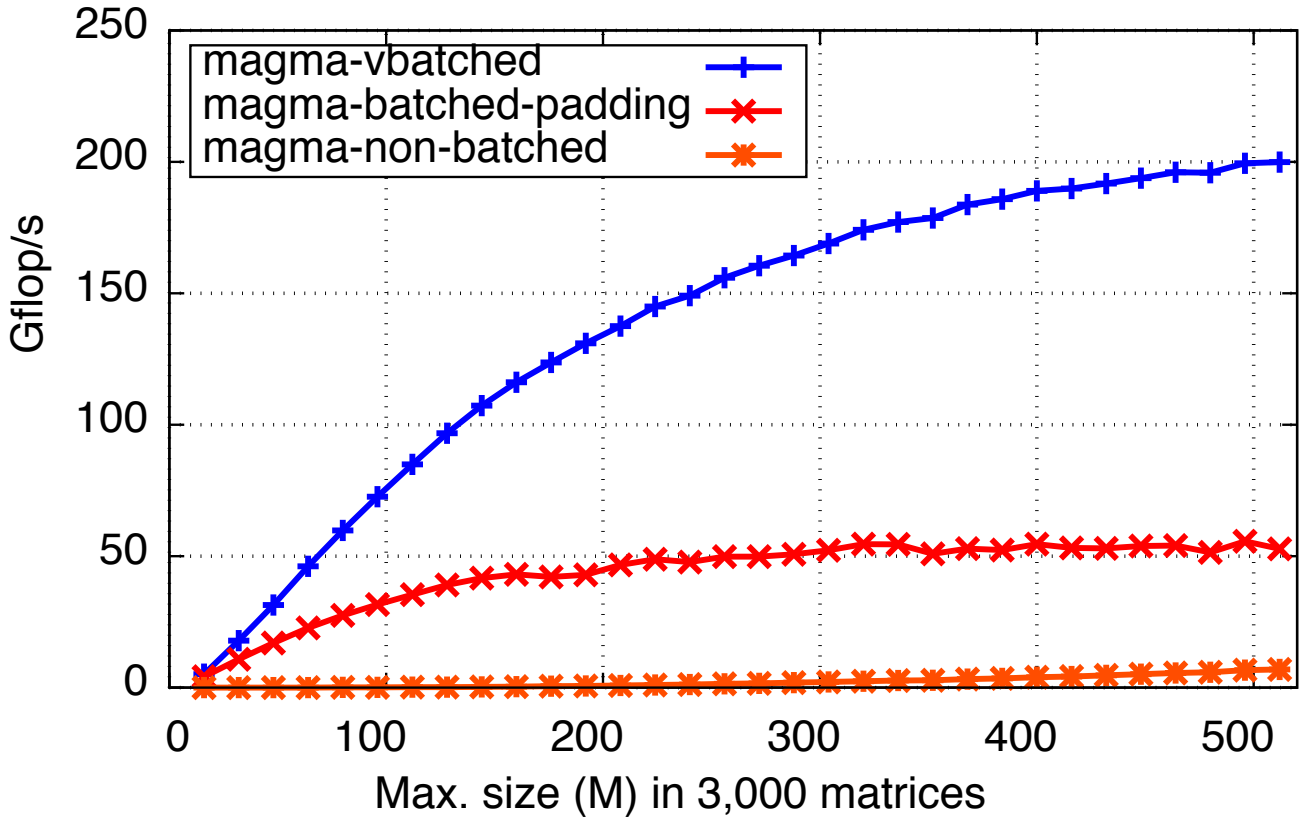
- Speedups from 7.9× up to 18.4×

Matrix-vector (DGEMV)



- Speedups from 4.4× up to 50+×

Cholesky Factorization (DPOTRF)



- Against padding: speedups from 1.24× up to 3.8×
- Against hybrid: speedups from 28× up to 200+×

Outline

Conclusion and Future Work

- Conclusion
  - Variable size batched computations can run efficiently on the GPU
  - New techniques are required
  - Autotuning is challenging (sizes, precision, batch count, distribution)
- What's Next?
  - Complete the set of BLAS routines
  - More LAPACK algorithms (LU and QR)
  - Test against different size distributions
  - Error reporting

Thank You!