Batched One-sided Factorizations on Hardware Accelerators Based on GPUs

Tingxing(Tim) Dong

Innovative Computing Laboratory
University of Tennessee, Knoxville

May 8th, 2015

Outline

- Motivations
- Algorithms and analysis
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

Motivations

- High-order FEMs,

  100x100, 200x200, batched GEMM, GEMV
- Astrophysics, subsurface transportation simulation,

  140x140 batched LU
- Radar signal processing,

  200x200 batched QR
- Computer vision

  batched Cholesky
- Accelerating multifrontal solvers/preconditioners for HSS matrices
- Further accelerating CA sparse iterative solvers
  (with a new mixed-precision orthgonalization technique)

Definition of batched LA

1. many small size matrices

   size: How small is small?

2. usually of the same size

   if not?

   -padding

   -multiple batches, the same sized batched together

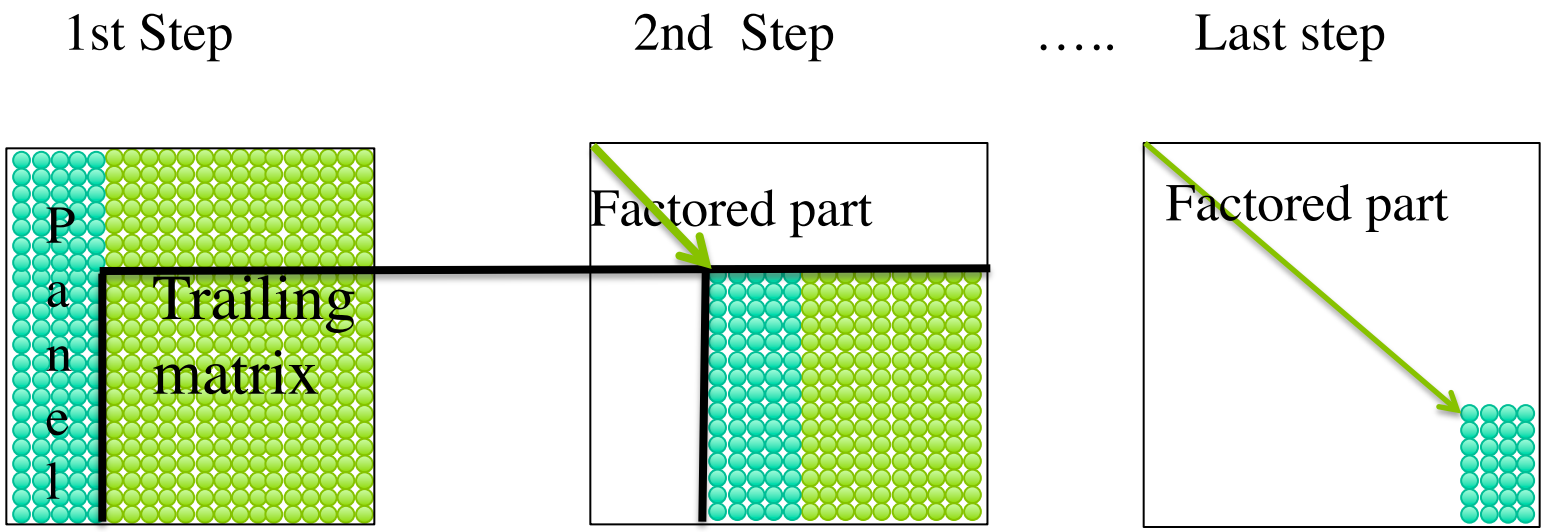
3 independent solved

4 processed together

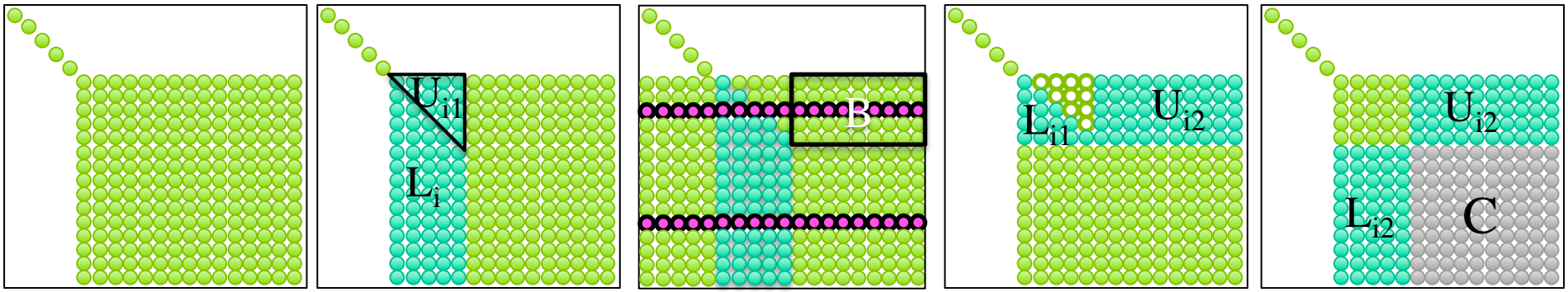
Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

Algorithm: a simplified overview



The LU factorization (details)



panel factorization
$PA_i = L_iU_{i1}$
dgetf2

swapping rows

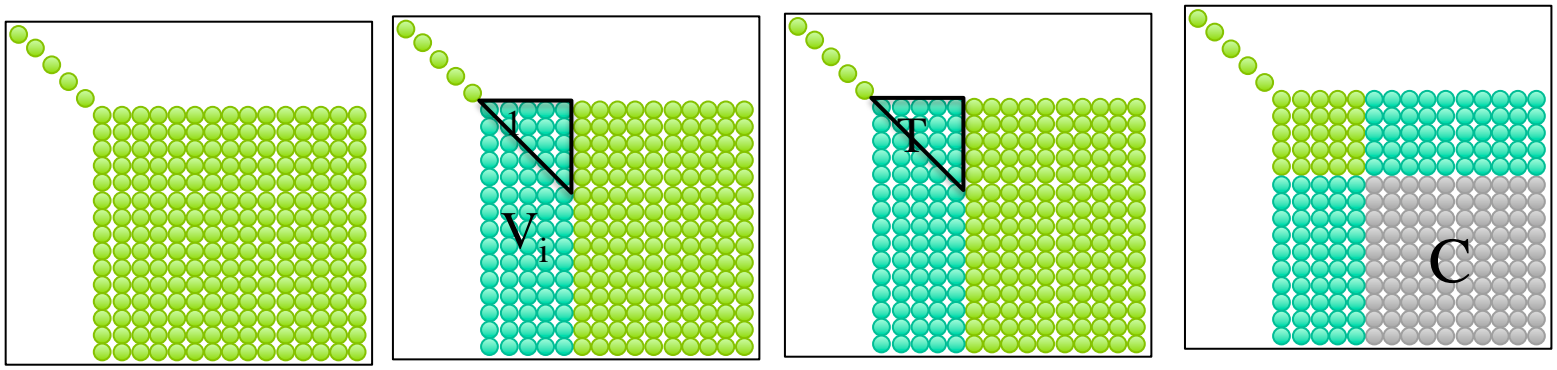
dlaswp

Triangular
update $U_{i2}=L_{i1}^{-1}B$
dtrsm

Schur update
$C = C - L_{i2}\,U_{i2}$
dgemm

- Panel factorization memory bound
- Triangular solve has little parallelism
- Schur complement (trailing matrix) update is the only easy task
- Performance killer: Partial pivoting

The QR factorization (details)



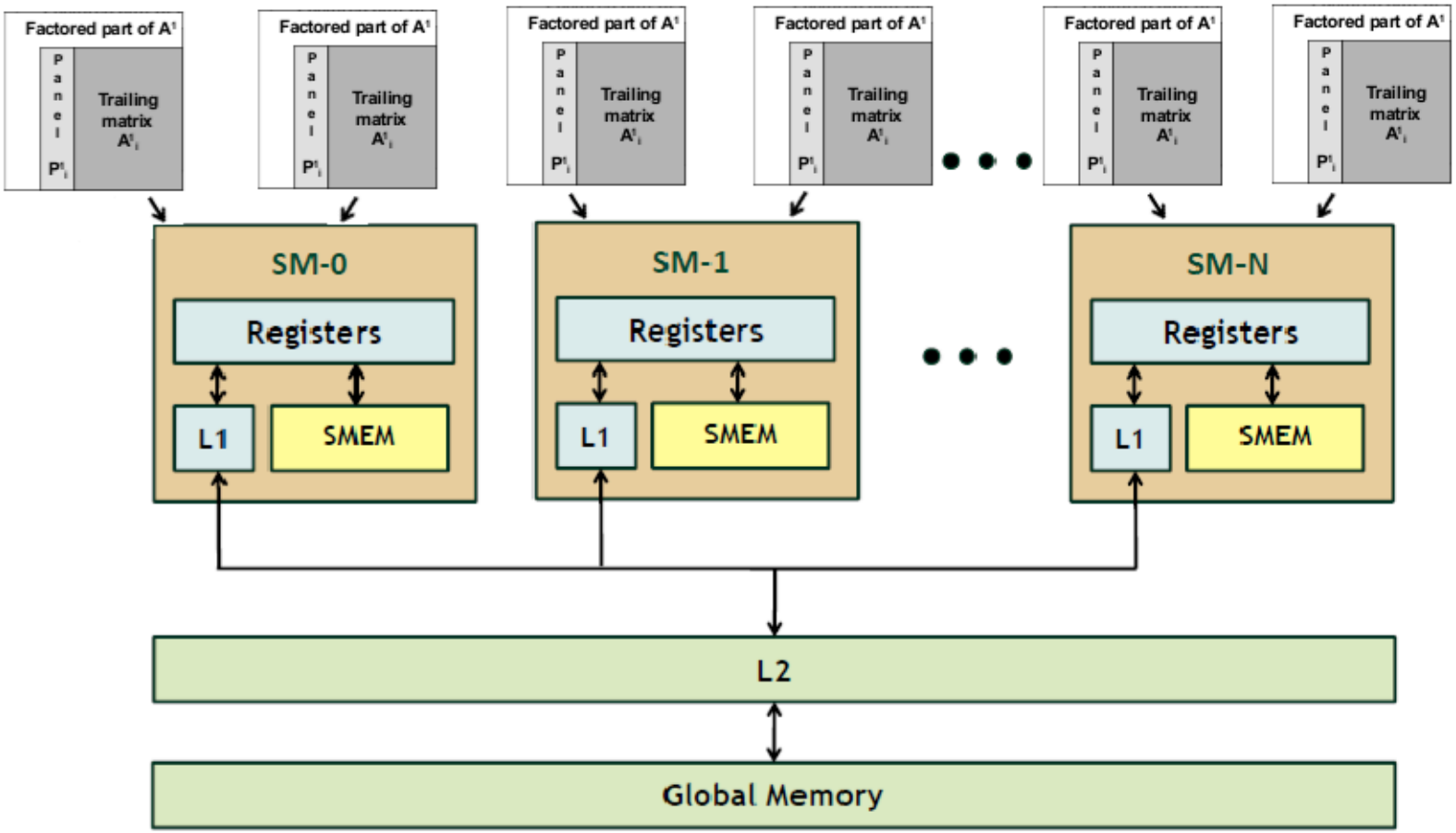
panel factorization
dgeqr2

Triangular
dlarft

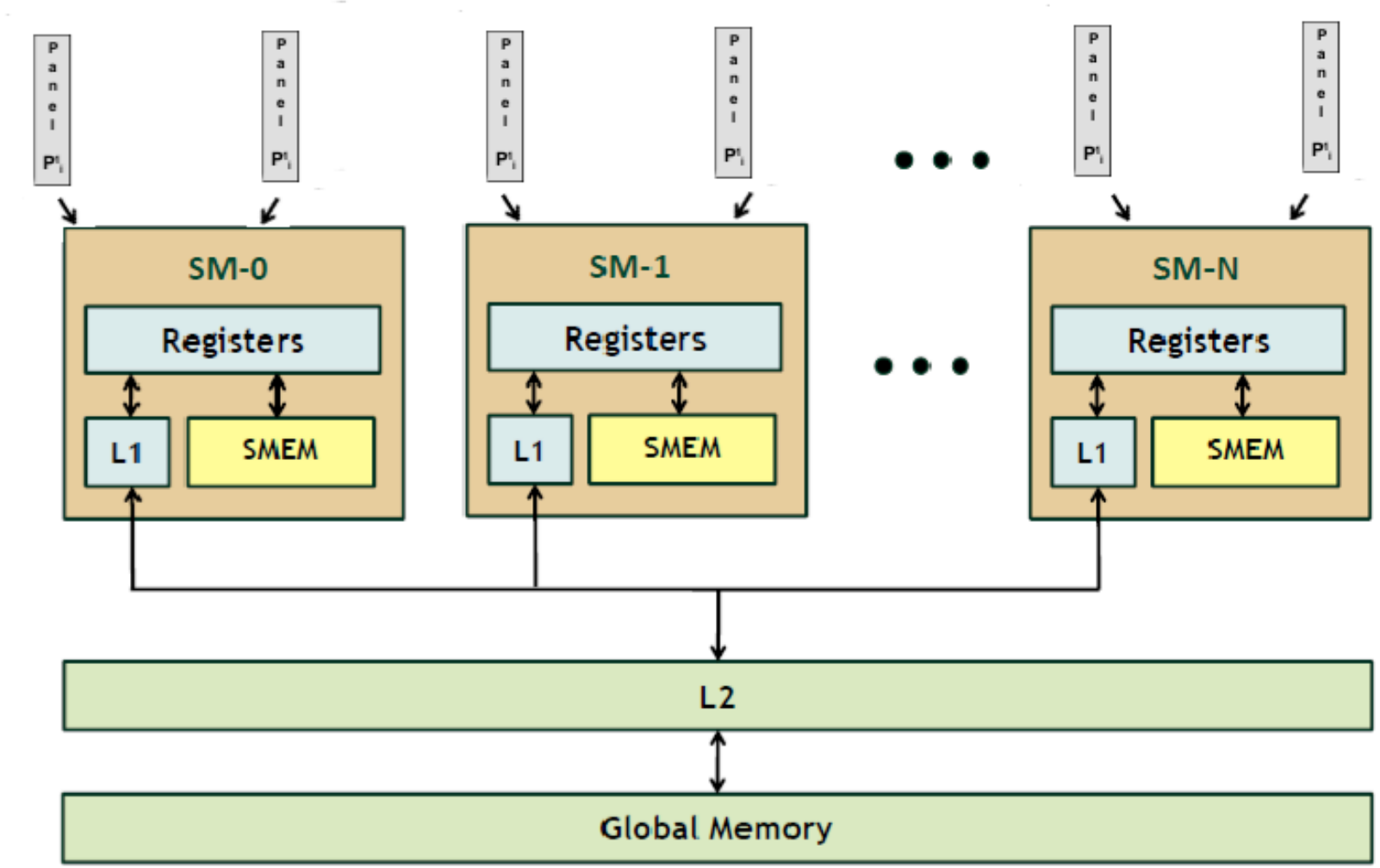
Schur update
C = (I- VT*V*)C
dlarfb(dgemm)

- Panel factorization: memory bound
- Triangular solve has little parallelism
- Schur complement (trailing matrix) update is the only easy task

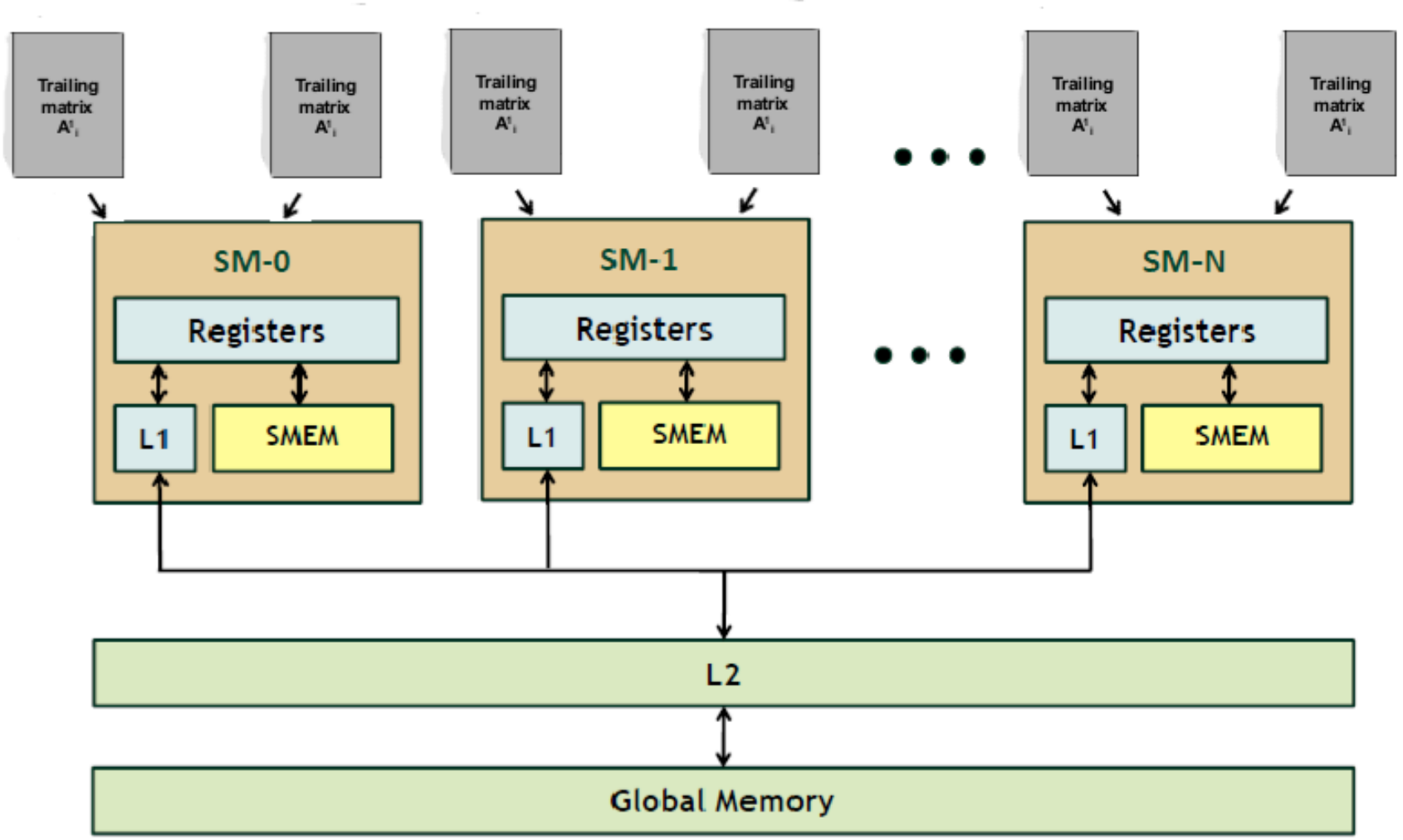
On GPU: a number of matrices distributed on the SMs



Batched panel factorization

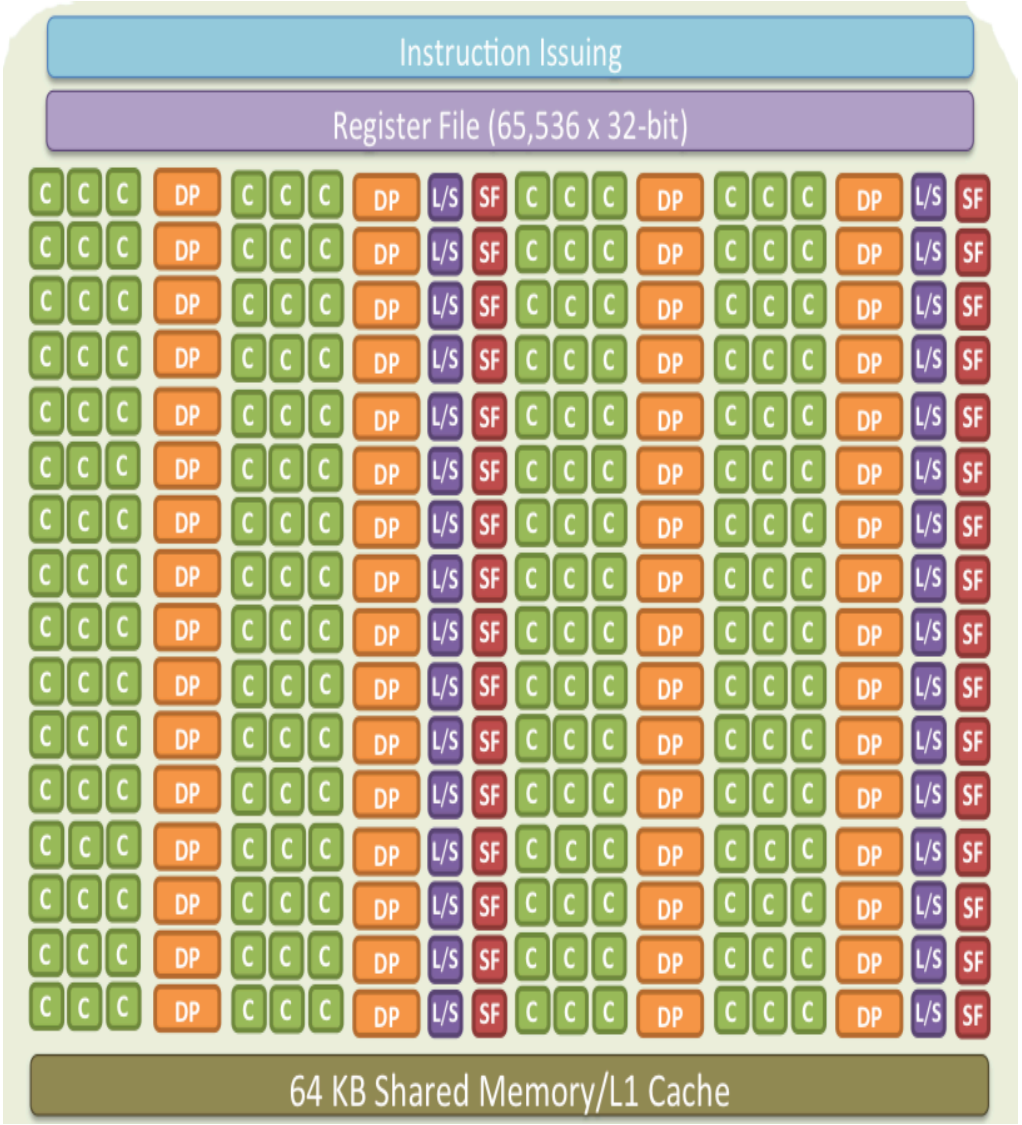


Batched trailing matrix update



CUDA Programming model: SIMD(T)

- SIMD(T): Single Instruction, multiple data (threads)
- SIMT: NVIDIA call it SIMT



Target matrix size

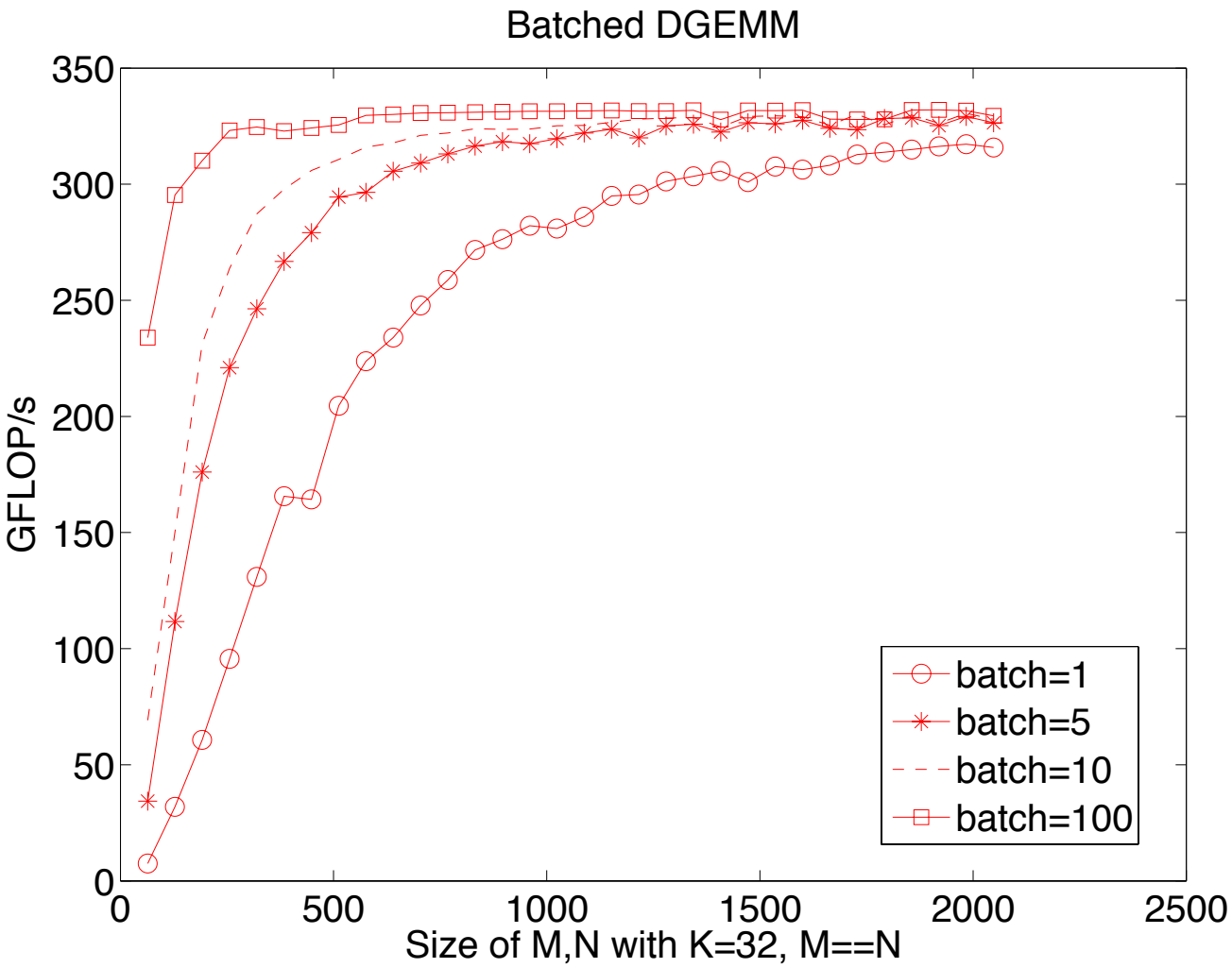
- Main consideration

  -GPU is saturated at certain matrix size.

  -Increasing number of matrices will lead to sequential processing

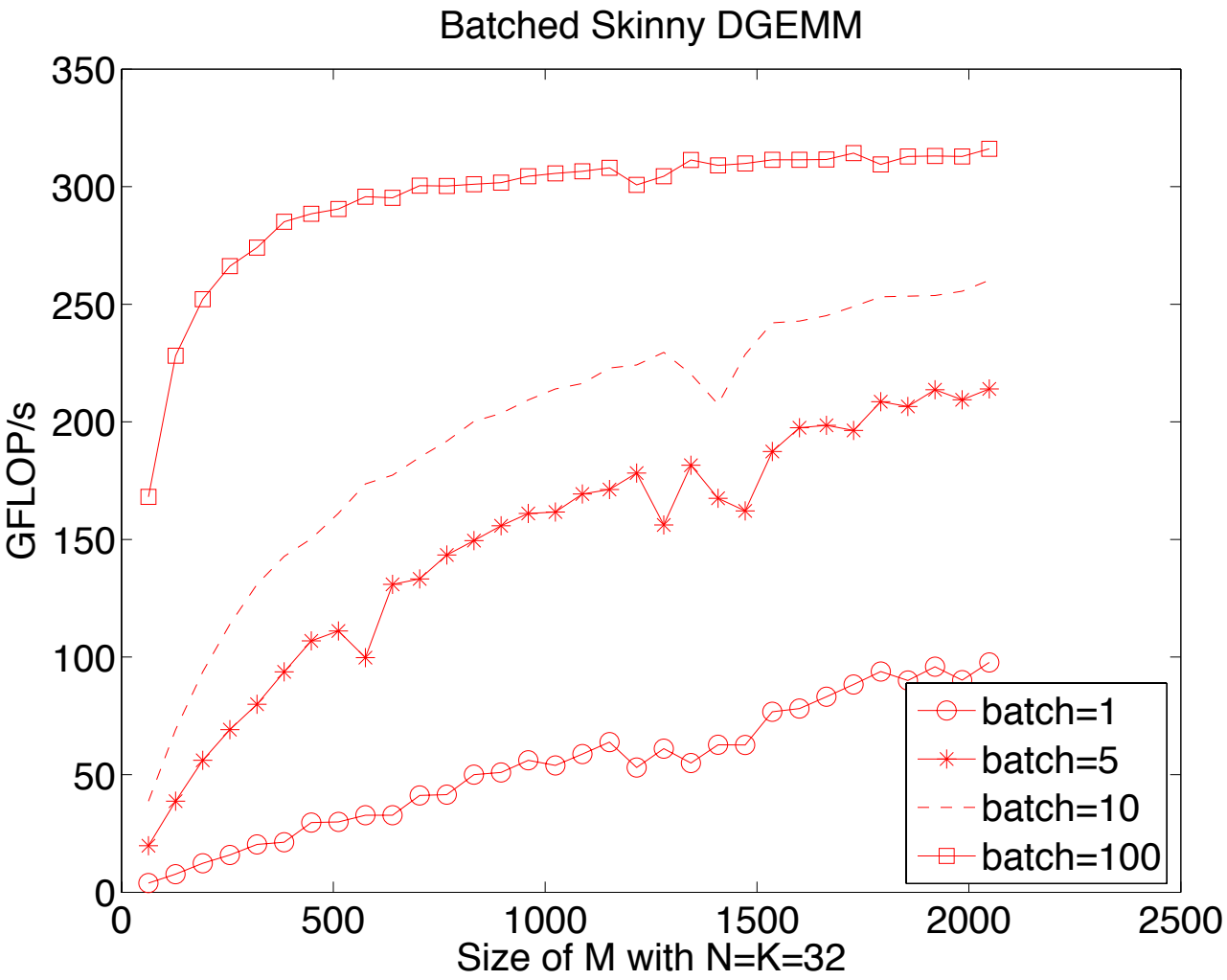
Trailing matrix behavior:

GEMM: two skinny matrix of the same size (A, B) produce a square matrix C



Panel behavior:

GEMM: a skinny matrix A multiply a small square one B produce a skinny matrix C



Target matrix size

- Maximum size

  square: 512x512

  skinny(or fat): 1024, (big enough)

- Beyond 1024, still produce correct result but not optimized
- classic MAGMA hybrid routine

Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

Building blocks: Batched BLAS

- Batched Level 1 BLAS

  xSCAL
- Batched Level 2 BLAS

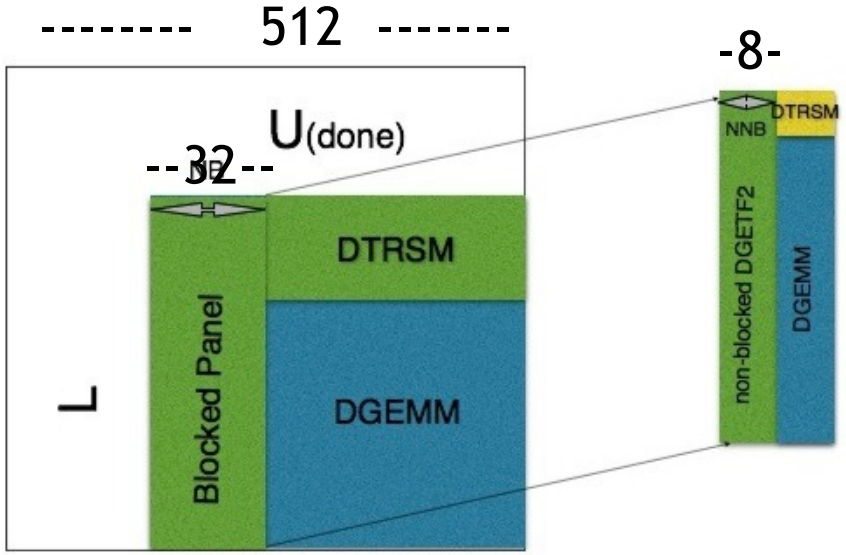
  xGER, xGEMV,
- Batched Level 3 BLAS

  xHERK, xTRSM, xGEMM/stream xGEMM
- Batched xGETF2, xPOTF2, xGEQR2
- Batched xLARFG, xLARF, xLARFT
- Batched solvers

Optimizations

- General Principles
  - ---Recursive blocking
  - ---Streamed GEMM
- Cholesky
  - ---Left-looking, RL variants (15% improvement)
- LU
  - ---Parallel swapping (50% improvement)
- QR
  - ---Triangular solver: T (30% improvement)

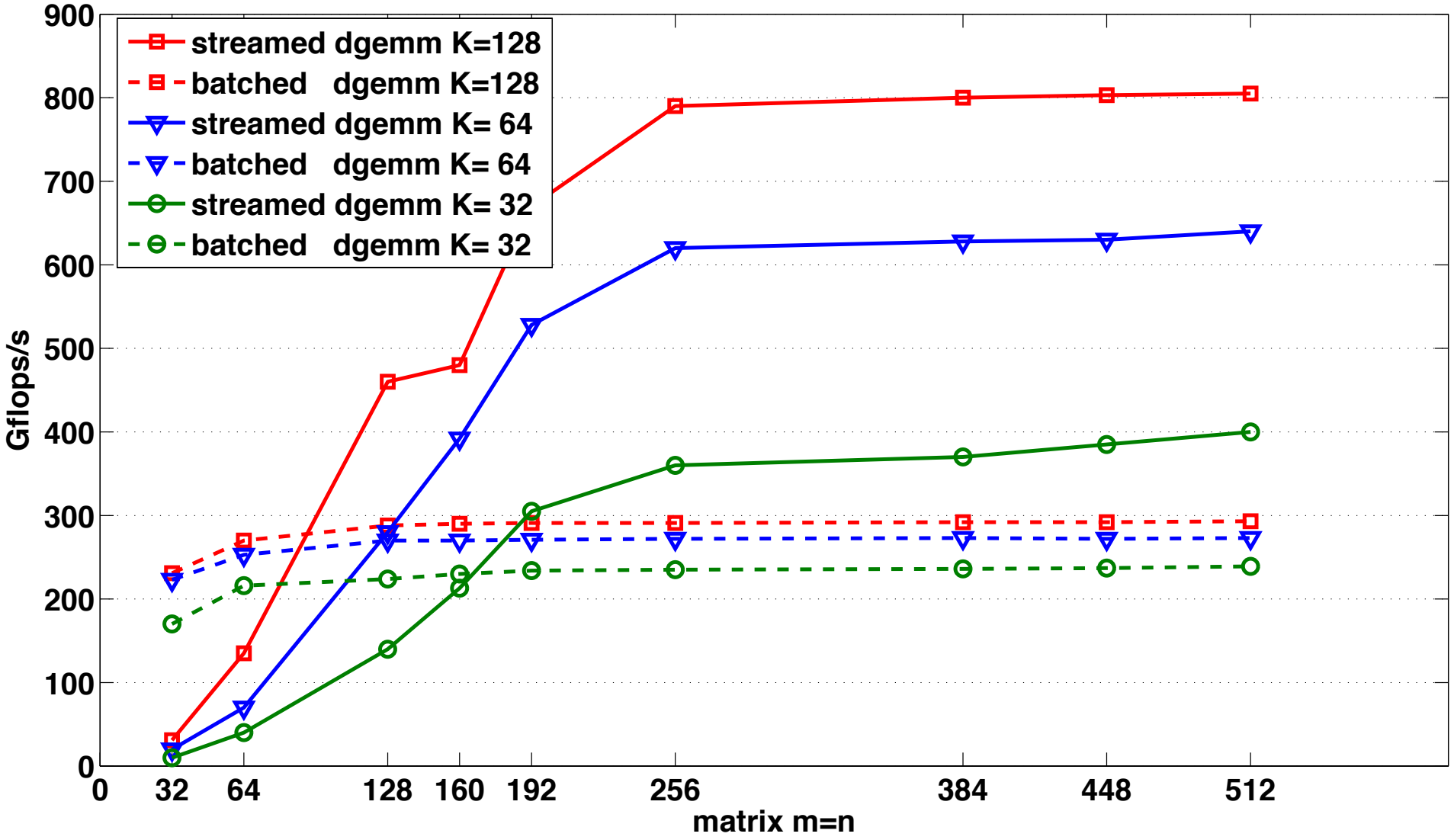
Recursive blocking:



Proposition:

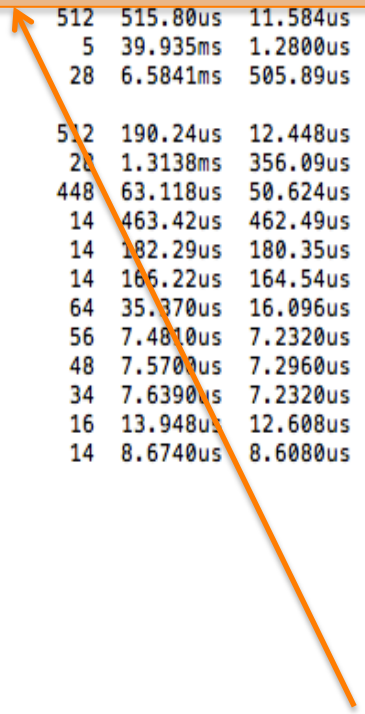
- **Do Nested blocking**:
  Develop a nested blocking technique that block also the panel in order to replace the dger kernel by dgemm kernel.

DGEMM Performance



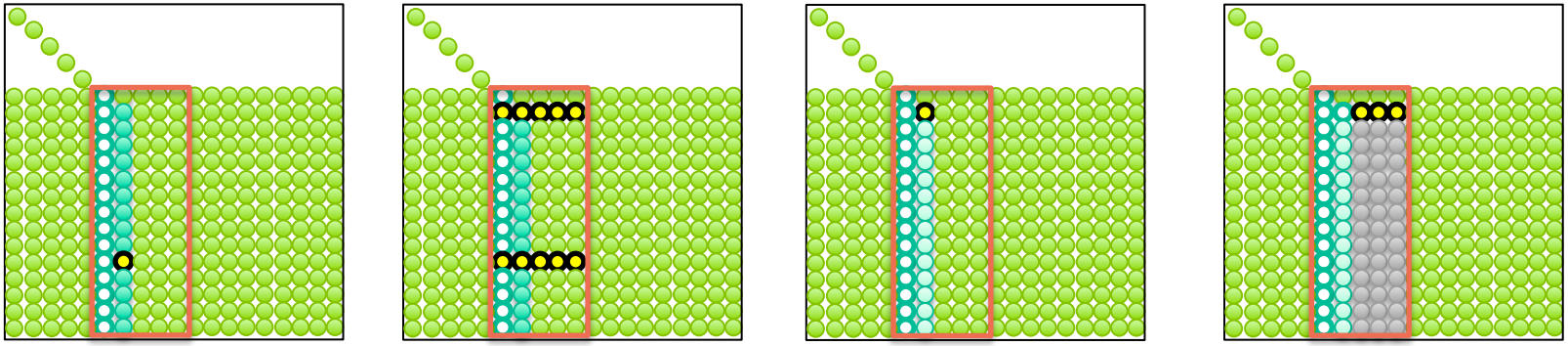
The killer in LU: serial swap

```
37.31%  928.92ms         4  232.23ms  211.04us  573.00ms  [CUDA memcpy DtoH]
 29.53%  735.31ms        28  26.261ms  5.5867ms  48.661ms  dlaswp_batched_kernel(int, double**, int, int, int, int**)
 10.61%  264.09ms       512  515.80us  11.584us  1.6749ms  kernel_dscal_dger(int, int, double**, int, int)
  8.02%  199.67ms         5  39.935ms  1.2800us  100.09ms  [CUDA memcpy HtoD]
  7.40%  184.36ms        28  6.5841ms  505.89us  27.352ms  void fermiPlusDgemmLDS128_batched<bool=0, bool=0, int=4, int=4, int=4, int=3, int=3>(double
ouble, double, int)
  3.91%  97.403ms       512  190.24us  12.448us  226.08us  kernel_dswap(int, double**, int, int, int**)
  1.48%  36.786ms        28  1.3138ms  356.09us  2.2575ms  dtrsmbatched_copy_kernel(int, int, double**, int, double**, int)
  1.14%  28.277ms       448  63.118us  50.624us  75.200us  kernel_idamax(int, int, double**, int, int, int, int**)
  0.26%  6.4878ms        14  463.42us  462.49us  464.19us  diag_dtrtri_kernel_lower(magma_diag_t, double**, double**, int)
  0.10%  2.5520ms        14  182.29us  180.35us  184.54us  triple_dgemm_update_16_part1_L(double**, double**, int, int, int)
  0.09%  2.3271ms        14  166.22us  164.54us  167.65us  triple_dgemm_update_16_part2_L(double**, double**, int, int, int)
  0.09%  2.2637ms        64  35.370us  16.096us  52.800us  kernel_idamax_2(int, double**, int, int, int, int**)
  0.02%  418.94us        56  7.4810us  7.2320us  8.1920us  kernel_dtrsm_set_pointer(double*, double**, int, int)
  0.01%  363.39us        48  7.5700us  7.2960us  7.7440us  kernel_set_A(double**, double*, int, int, int, int)
  0.01%  259.74us        34  7.6390us  7.2320us  7.8400us  kernel_set_ipiv(int**, int*, int, int)
  0.01%  223.17us        16  13.948us  12.608us  14.336us  Adjust_ipiv(int**, int, int)
  0.00%  121.44us        14  8.6740us  8.6080us  8.7360us  kernel_set_pointer(double**, double**, double**, double*, int, int, int, int)
```



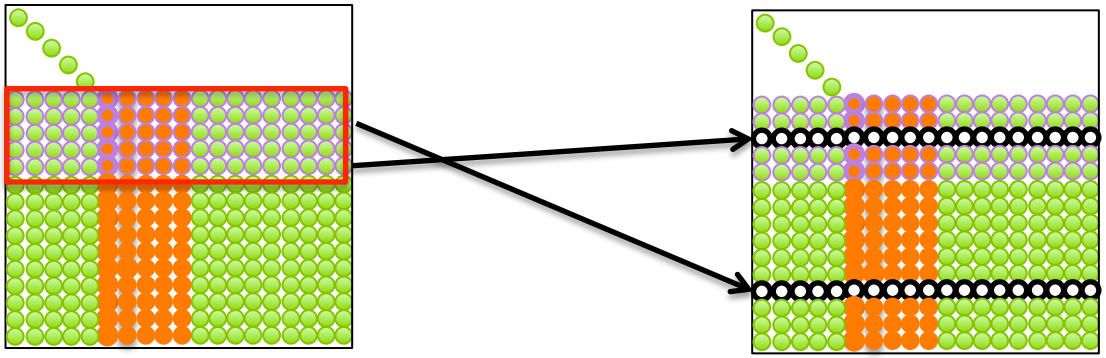
60% of the time

Serial swapping in panel:



Record their final destination row

Parallel swapping in trailing matrix



Triangular solver T in QR

- LAPACK algorithm

  ---BLAS-2 routines: GEMV, TRMV

  results in memory un-coalesced access

- Batched:

  -- GEMM, TRMV(in shared memory)

  -- 30% improvement

Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

Testing hardware

- CPU: two sockets of 8 core Intel Sandy Bridge E5-2670

  2.65GHz, 20MB L3,

  MKL 16 threads

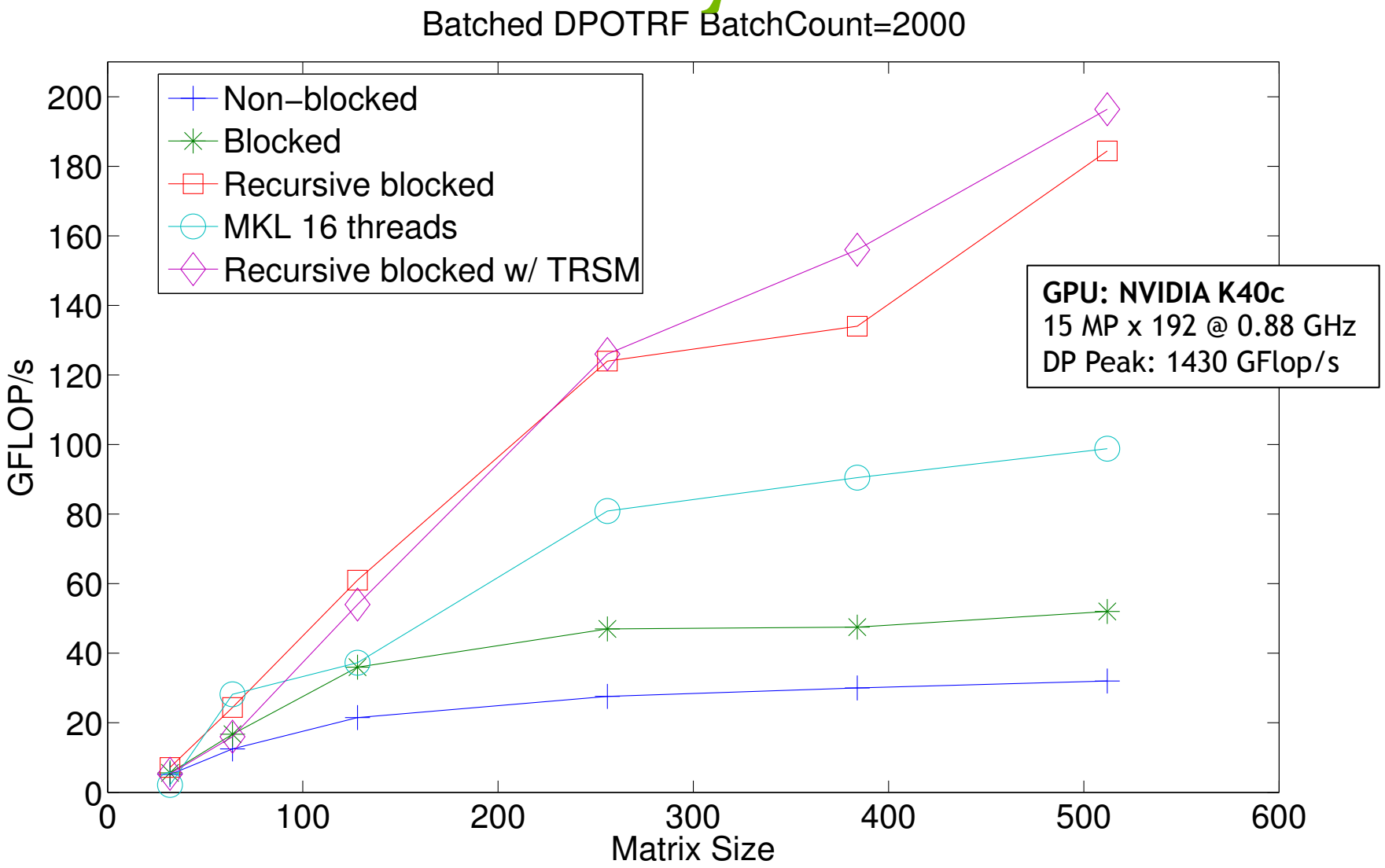
  TDP: 115W * 2

- GPU: NVIDIA K40c,

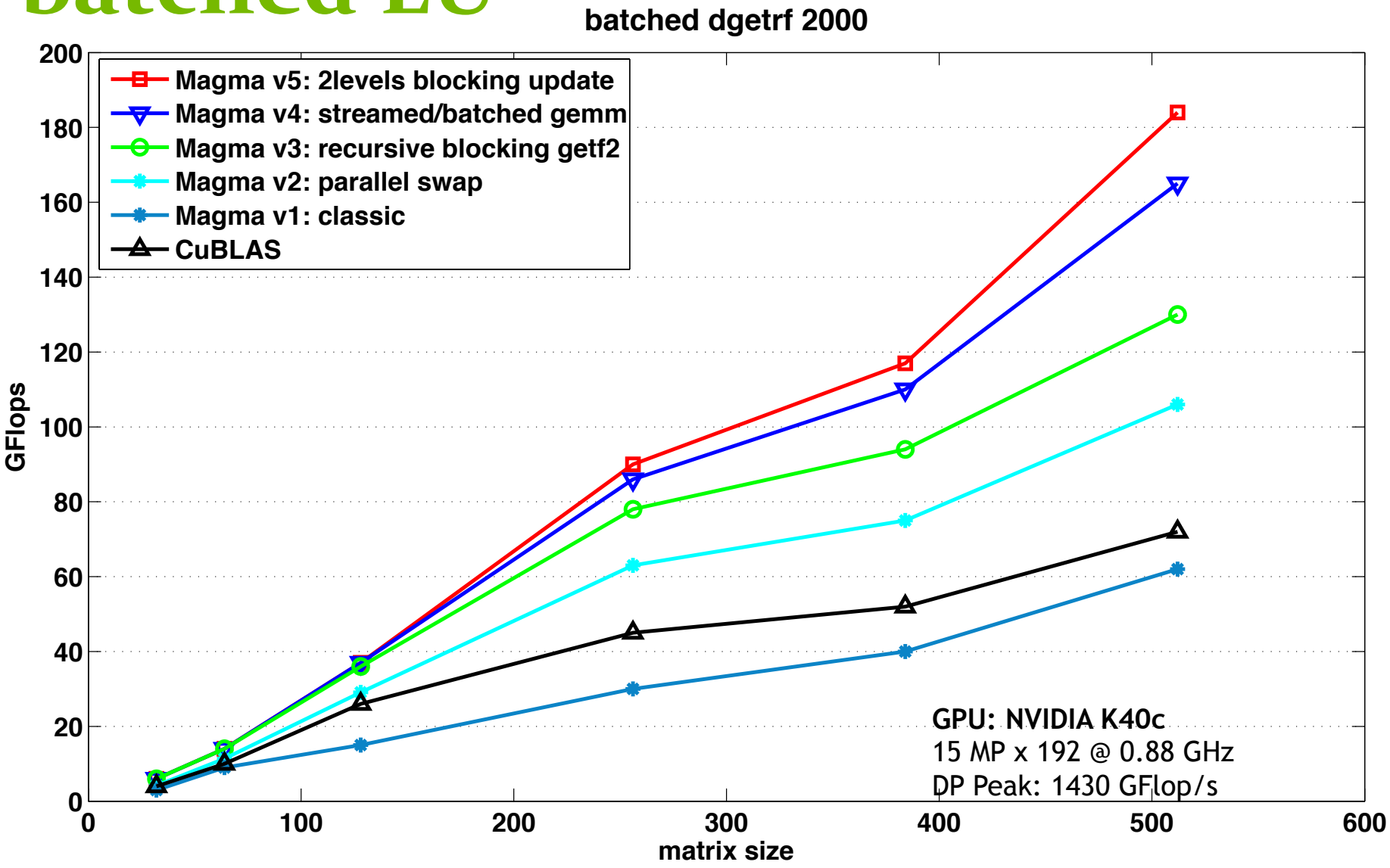
  2880 cores with 0.8GHz

  TDP : 235W

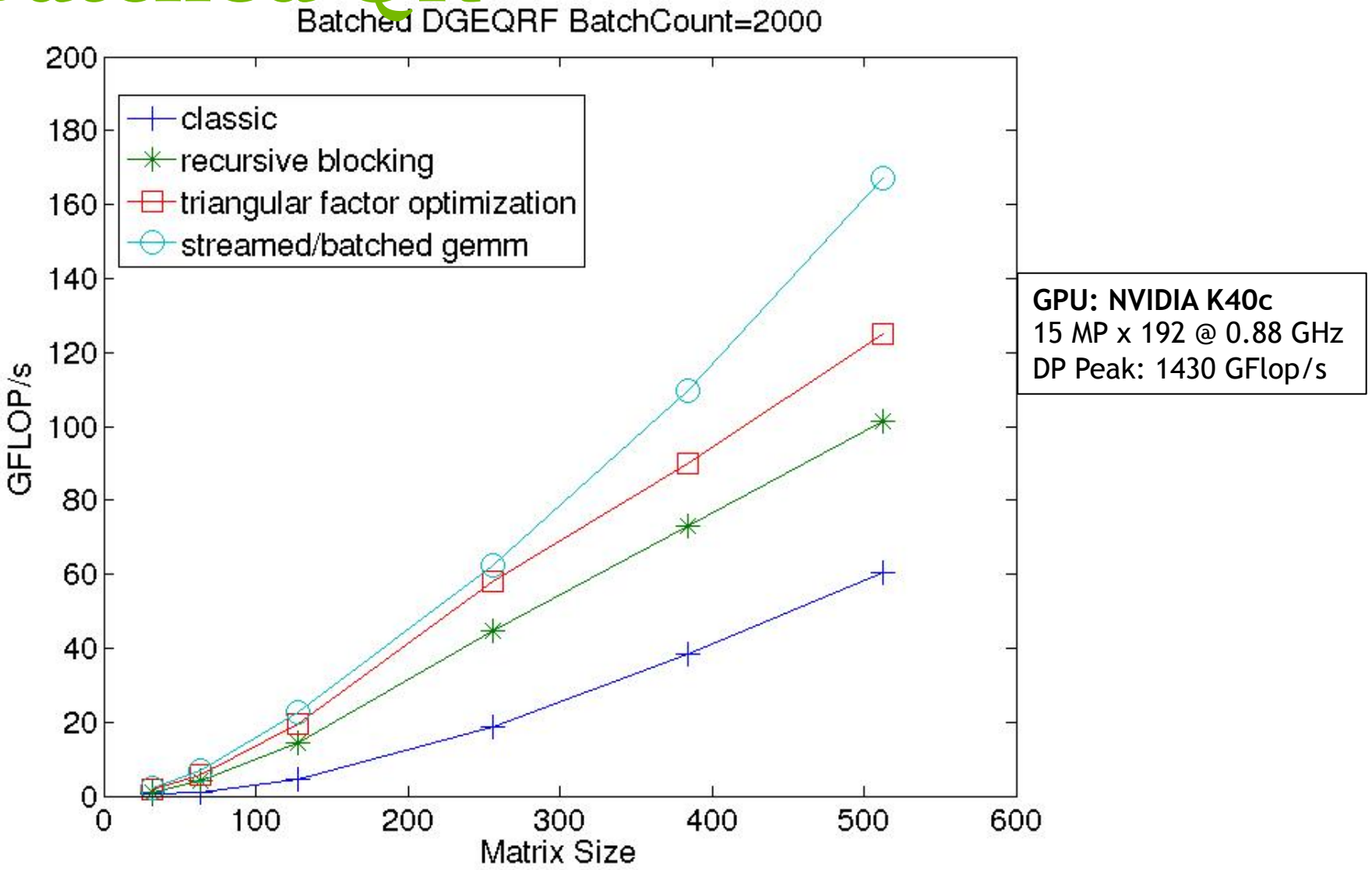
Batched Cholesky



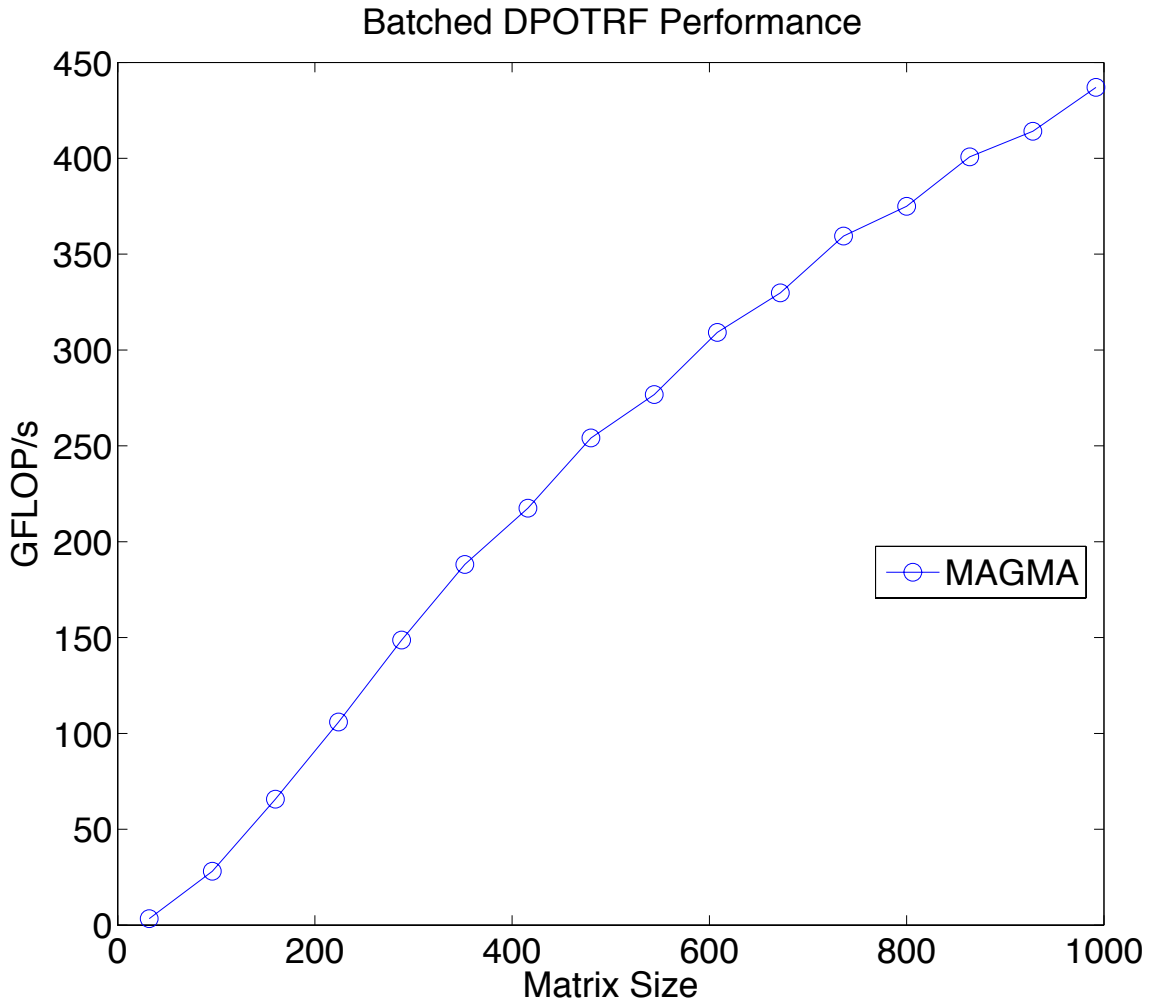
batched LU


batched dgetrf 2000
Magma v5: 2levels blocking update
Magma v4: streamed/batched gemm
Magma v3: recursive blocking getf2
Magma v2: parallel swap
Magma v1: classic
CuBLAS
GFlops
matrix size
0
20
40
60
80
100
120
140
160
180
200
0
100
200
300
400
500
600
GPU: NVIDIA K40c
15 MP x 192 @ 0.88 GHz
DP Peak: 1430 GFlop/s

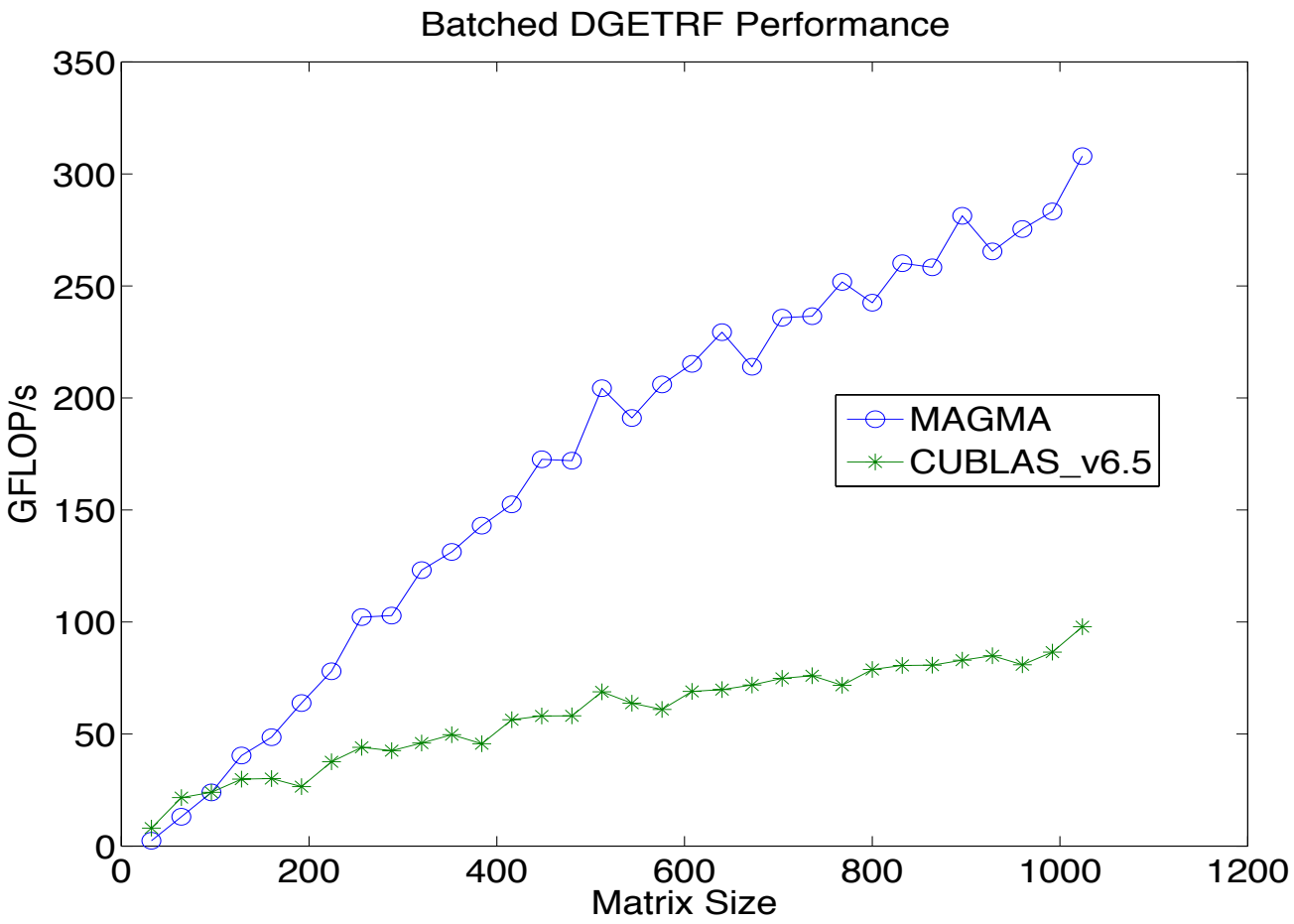
Batched QR


Batched DGEQRF BatchCount=2000
classic
recursive blocking
triangular factor optimization
streamed/batched gemm
GFLOP/s
Matrix Size
0
20
40
60
80
100
120
140
160
180
200
0
100
200
300
400
500
600
GPU: NVIDIA K40c
15 MP x 192 @ 0.88 GHz
DP Peak: 1430 GFlop/s

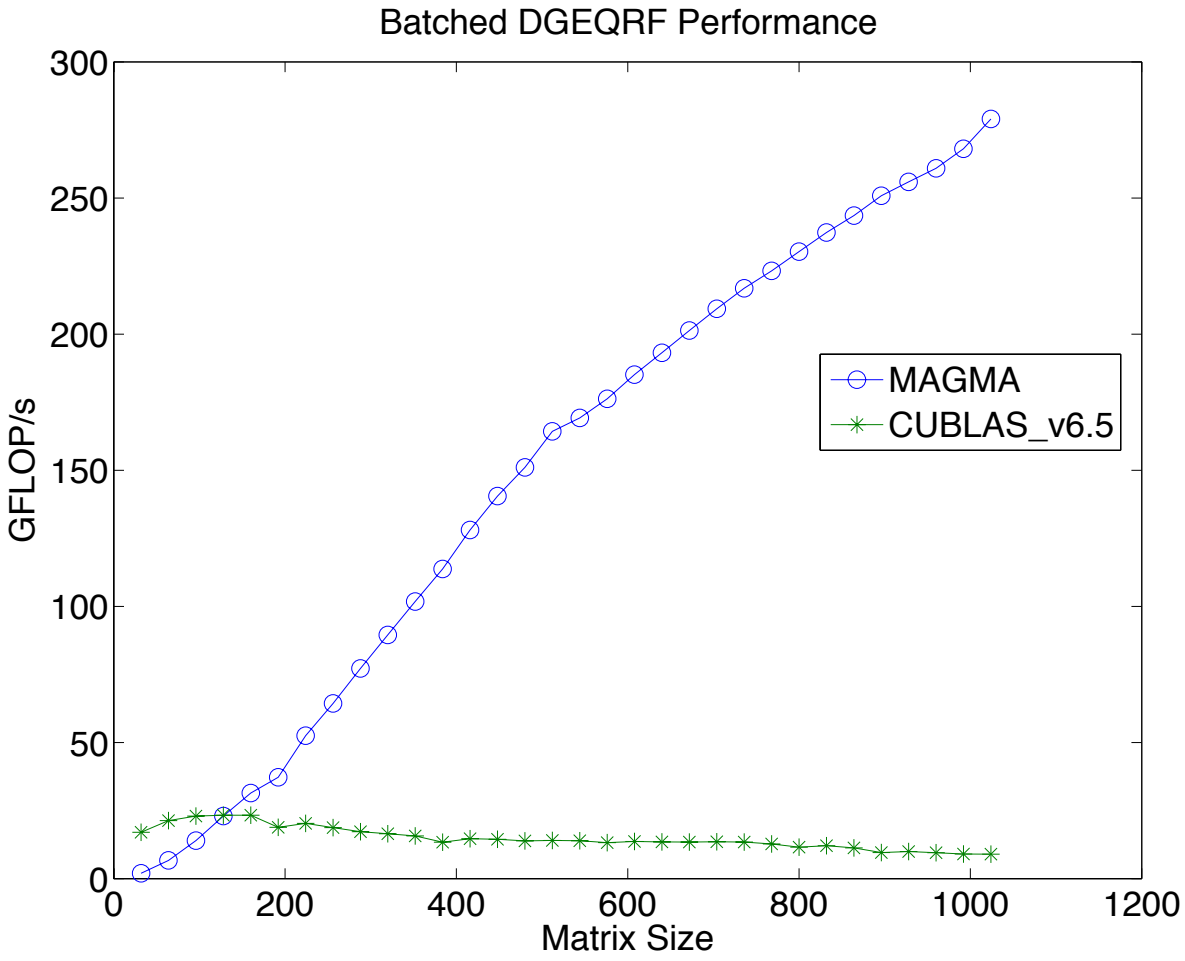
batched Cholesky (up to 1024)



batched LU(up to 1024)



batched QR (up to 1024)

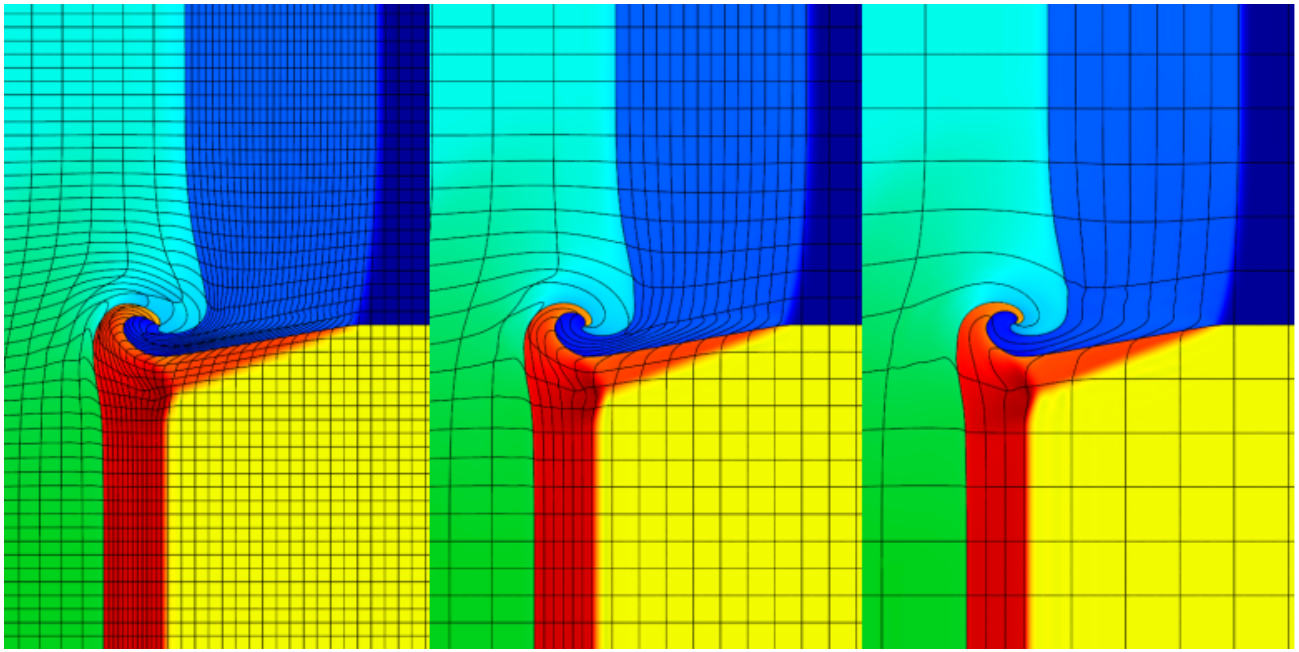


Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

High order method FEM

Q8-Q7	Q4-Q3	Q2-Q1



10,000 small matrix size:

- GEMM, GEMV
- Conjugate

order=1,2,3	Q1-Q0	Q2-Q1	Q3-Q2
$\hat{\phi}$	12 * 4	30 * 24	60 * 126
Jz	dim * dim (2 or 3)		

Another batched solution(solution 2):

```
__device__ void MultAtB{  };
__device__ void CalcAdjugate {  };
__device__ void SVD {  };

__global__ void  kernel_unroll_loop
{
    int tid = threadIdx.x ;
    tid call CalcAdjugate();
    tid call SVD();
    tid call MultAtB();
}
int main(){
            kernels_unroll_loop<<< threads, blocks >>>();
}
```

Another batched solution (solution 2):

- One thread solves one task (matrix problem)

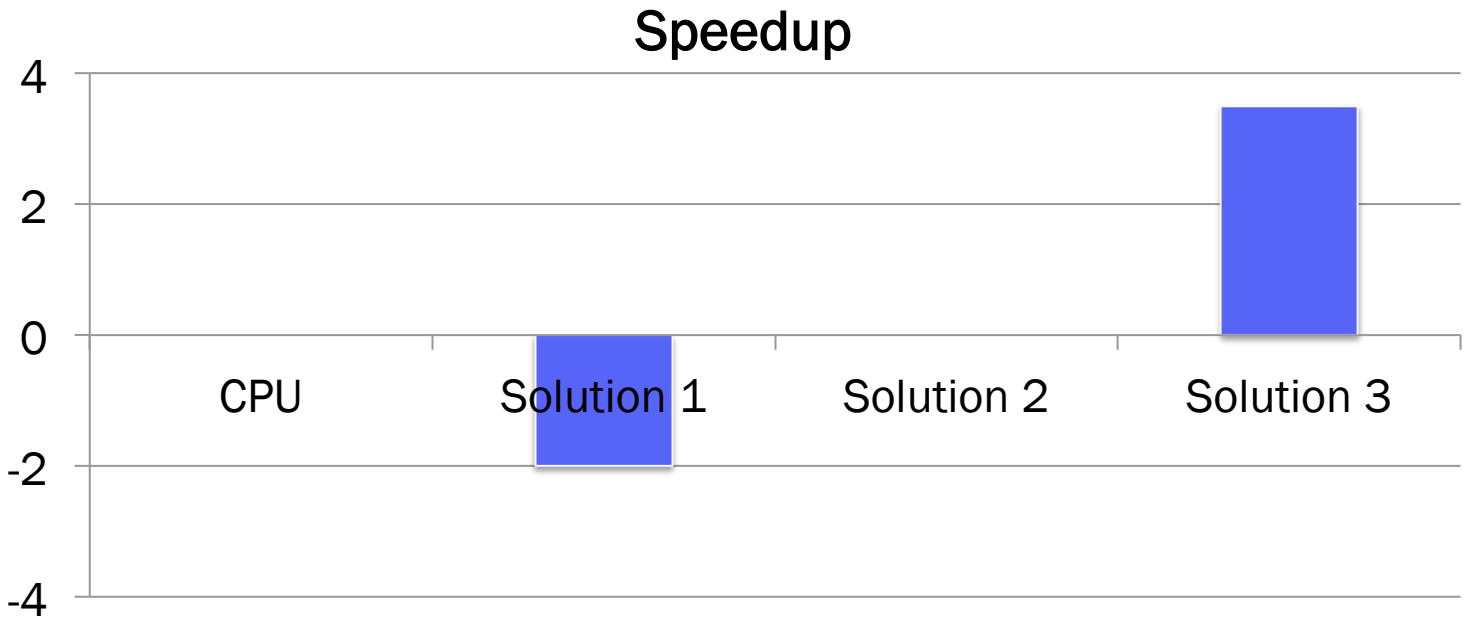
- This approach is also adopted by

V. Oreste, M.Fatica, N. A.Gawande, A.Tumeo, “Power/ performance trade-offs of small batched LU based solvers on GPU”s. *Euro-Par 2013*.

Batched BLAS solution (solution 3)

- Solution 1: non-batched call MAGMA individually
- Solution 2: batched w/o multi-threading BLAS
- Solution 3: our solution

Fermi GPU and 6 core CPU



The reason of low performance solution 2

- One thread access one matrix (e.g 30*24)

  -- memory non-coalesced

- The BLAS routines is parallel on task level

  -- inherently sequential not multi-threading

- GPU's SIMT is not exploited

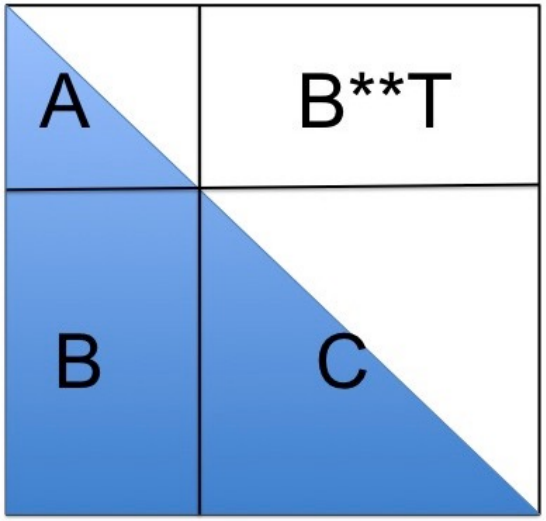
Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

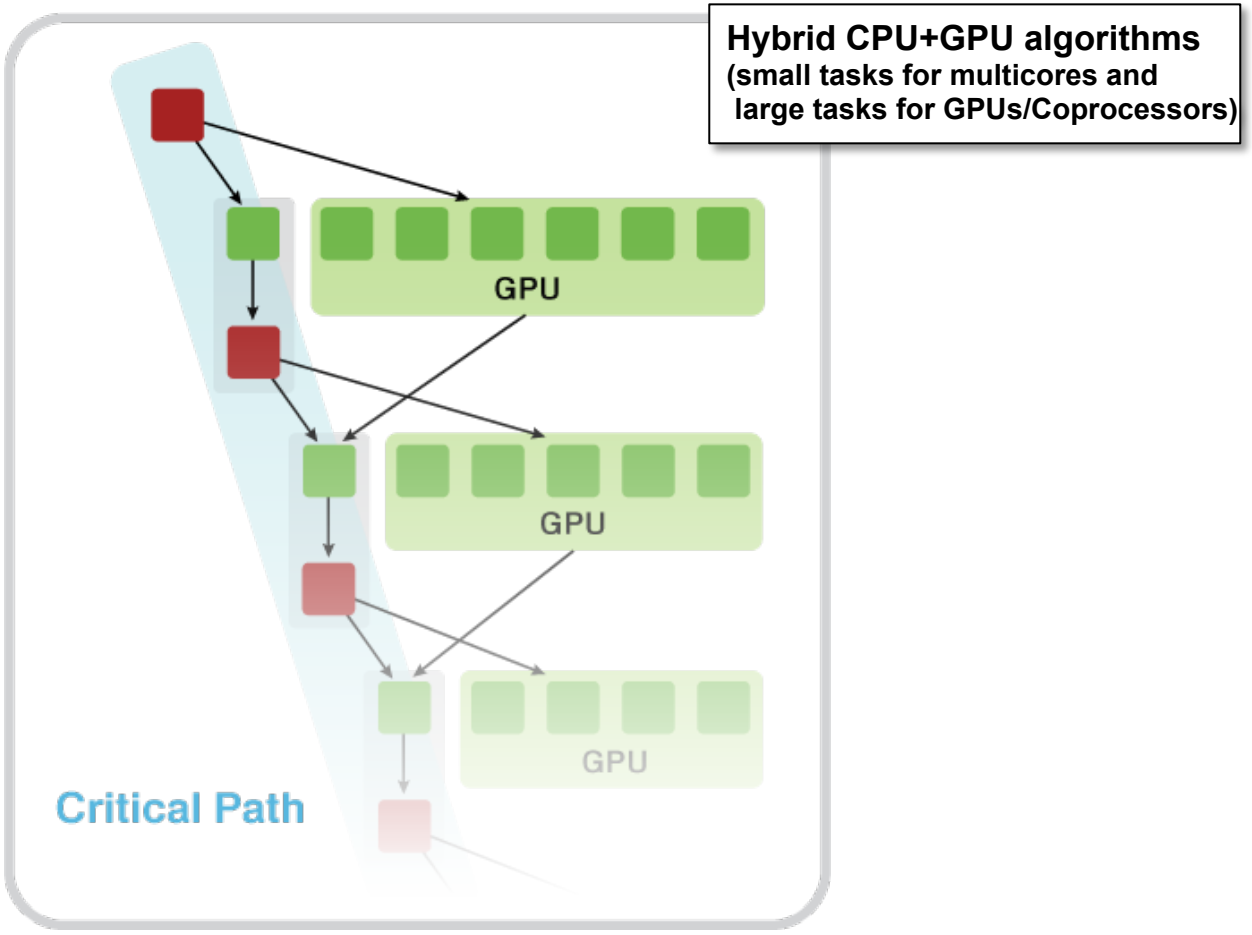
A native Cholesky target on large size

- Three solutions for large matrix size problem:
  -- MKL: CPU solution
  -- Hybrid MAGMA: A on CPU; B, C on GPU
  -- Native: A is factorized by batched implementation with one matrix

- B, C rely on the result of A
  -- A is critical path
- Size of A is : nb = 512



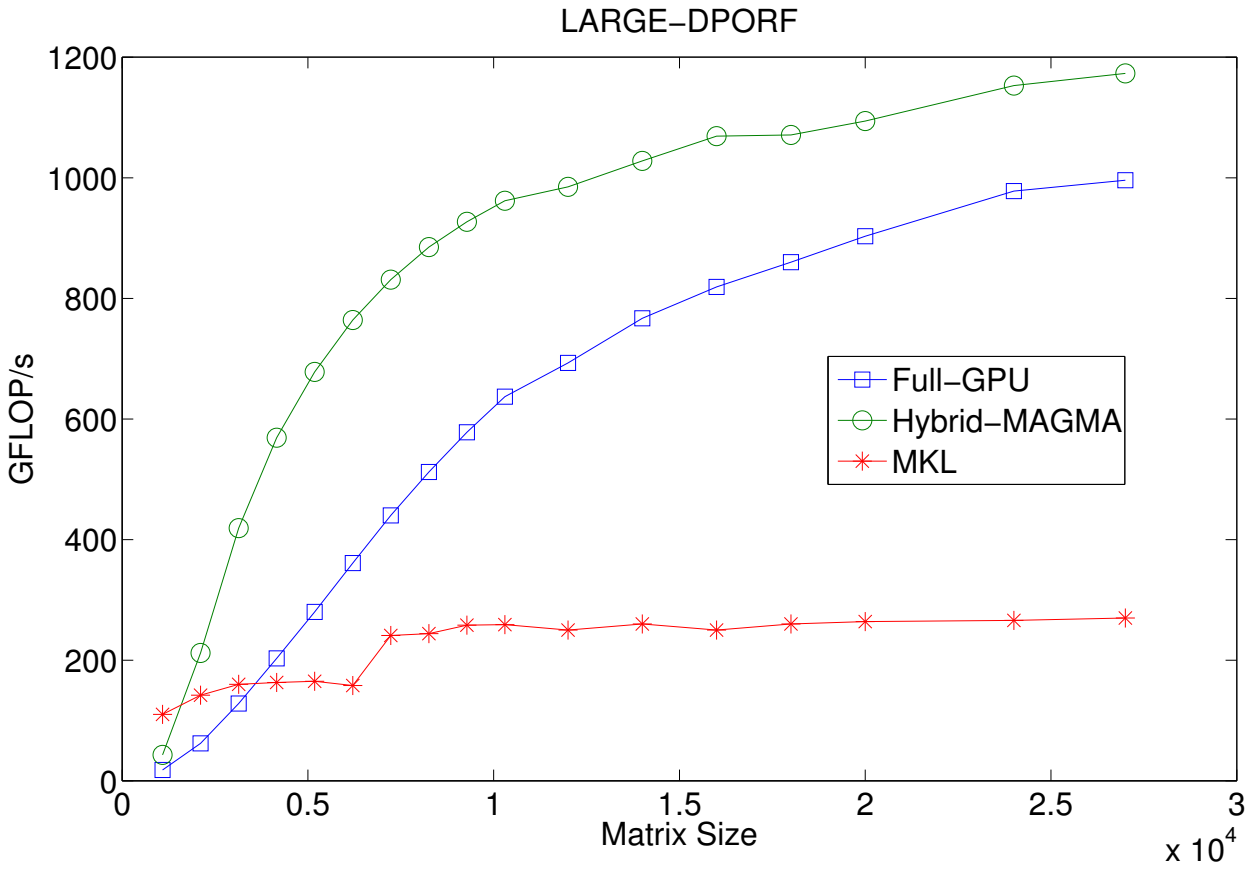
Tile A is on the critical path



A native Cholesky target on large size

Ratio of A = nb/N

= N/nb * (nb * nb)/ (N*N)

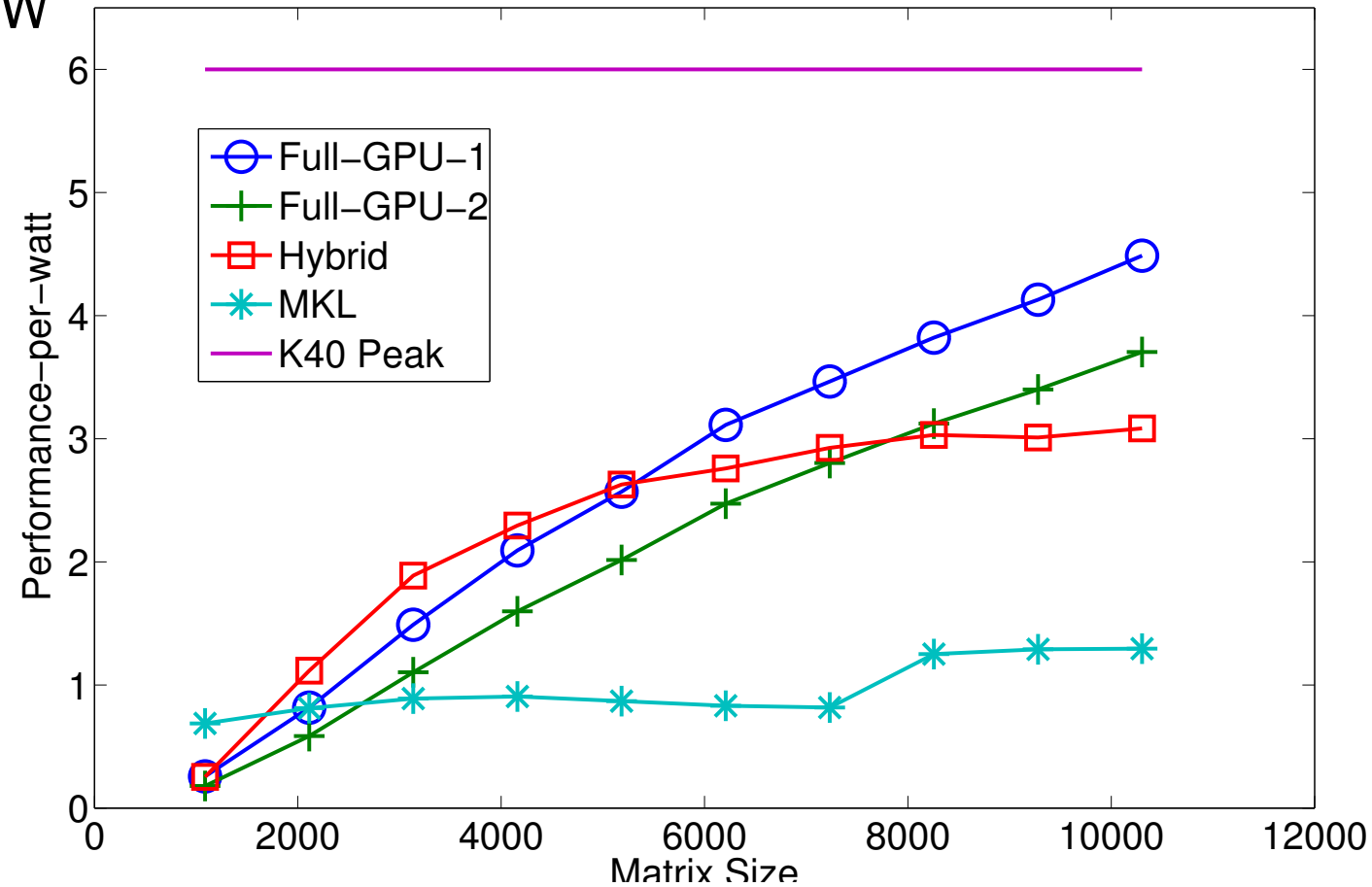


How good it is

Gflop/s	K40c	CPU	Percentage of dgemm
dgemm	1200	300	N/A
Hybrid Cholesky	1200		1200/1500= 75%
Native(Full-GPU)	1000	N/A	1000/1200 = 83%

A native Cholesky performance-per-watt

- Full-GPU-1: no CPU idle power considered
- Full-GPU-2: CPU idle power
- 4.5Gflops/W



Outline

- Motivations
- Algorithms
- Implementations and optimizations
- Performance results
- A case study: a CFD application
- Power
- Conclusions and future work

Conclusions(1/2)

- Batched LA on Accelerators is not well studied
  -- batched routines are recently added in CUBLAS
  -- Vendor library (CUBLAS) performance are slow

- Accelerators are all in SIMD architecture
  -- software must exploit the SIMT architecture
  -- our batched solution is based on batched BLAS
  -- multi-threaded BLAS to exploit the SIMT architecture

Conclusions(2/2)

- Optimizations require algorithmically change
  --not porting LAPACK
  --parallel swapping, triangular solver

- Batched problem is a building block for other problems
  --native large size problem
  --skinny QR: split into batches
  --sparse LA

Future work: bi-digonalization

- 1st Step of SVD which is a two-sided factorization
- Applied in many areas

Building blocks required

- Batched Level 2 BLAS

  xGEMV,
- Batched Level 3 BLAS

  xGEMM/stream xGEMM
- Batched xLARFG (householder),

thanks for your patience!