High-Performance Batched Computations

Azzam Haidar

Innovative Computing Laboratory
University of Tennessee,
Knoxville

In collaboration with:
LLNL, Livermore, CA, USA
University of Manchester, Manchester, UK
University of Paris-Sud, France

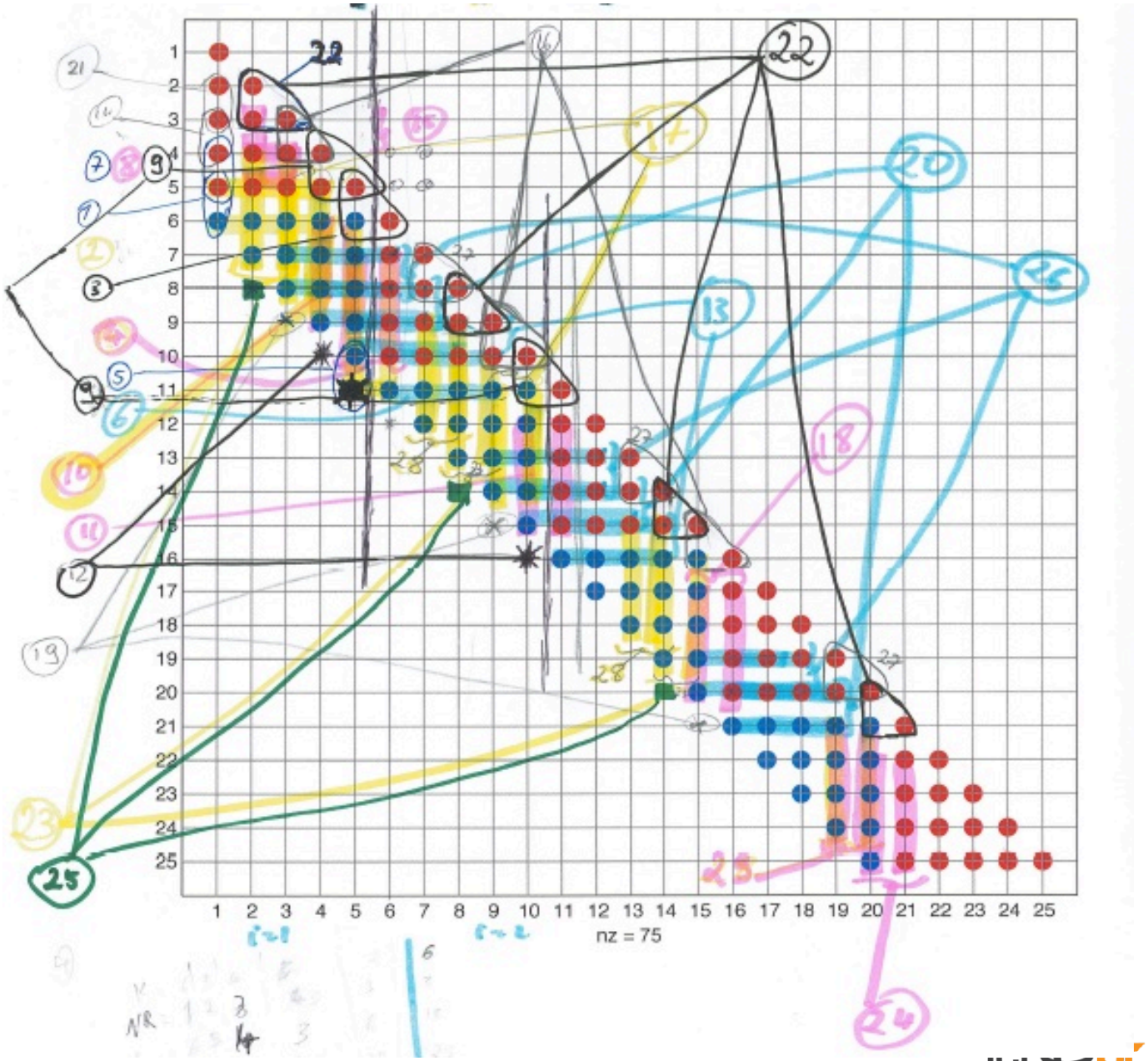
May 18–19, 2016

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Outline

- Motivation
- Current approaches and challenges
- **MAGMA Batched computations**
  - **Algorithmic basics**
  - **Design and optimizations for batched computations**
  - **LU, QR, and Cholesky**
  - **Performance results**
  - **Energy efficiency**
- MAGMA Batched computations for variable sizes
- **Future direction**

MAGMA Batched Computations



Batched computation

```
int main (int argc, char *argv[])
{
int nthreads, tid;
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
  {
  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf( »Batched computation from thread = %d\n", tid);
  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  }  /* All threads join master thread and disband */
}
```

MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Our goals:

- Develop a **high-performance** numerical library for **batched** linear algebra subroutines **tuned for performance and energy efficiency** on modern processor architectures, CPU, GPU, Phi
- Define **modular interfaces** that allow code replacement techniques [to provide the developers of applications, compilers, and runtime systems with the option of expressing new, application-specific batched computations]
- Propose **template** design and **code auto generation** for performance portability

MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

- There is a **lack of HPC linear algebra software for small problems** especially for GPU
- **CPU**: this can be done easily using existing software infrastructure
  1. naïve way: what we call Non-batched design
  2. better way: Batched design

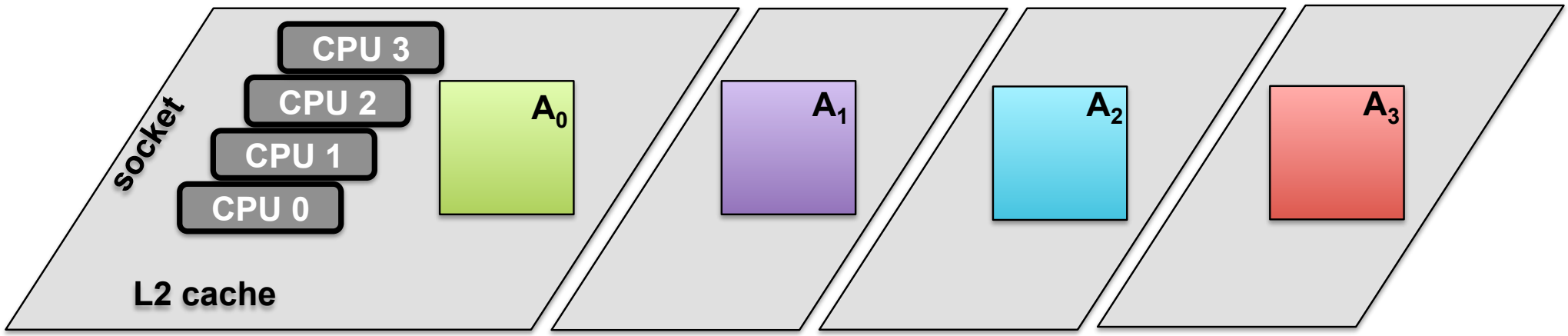
MAGMA Batched Computations CPU

1. Non-batched computation

loop over the matrices one by one and compute either:

- sequentially wasting all the other cores, and attaining very poor performance
- Or using multithread (note that for small matrices there is not enough work for all cores so expect low efficiency as well as threads contention can affect the performance)

```
for (i=0; i<batchount; i++)
```



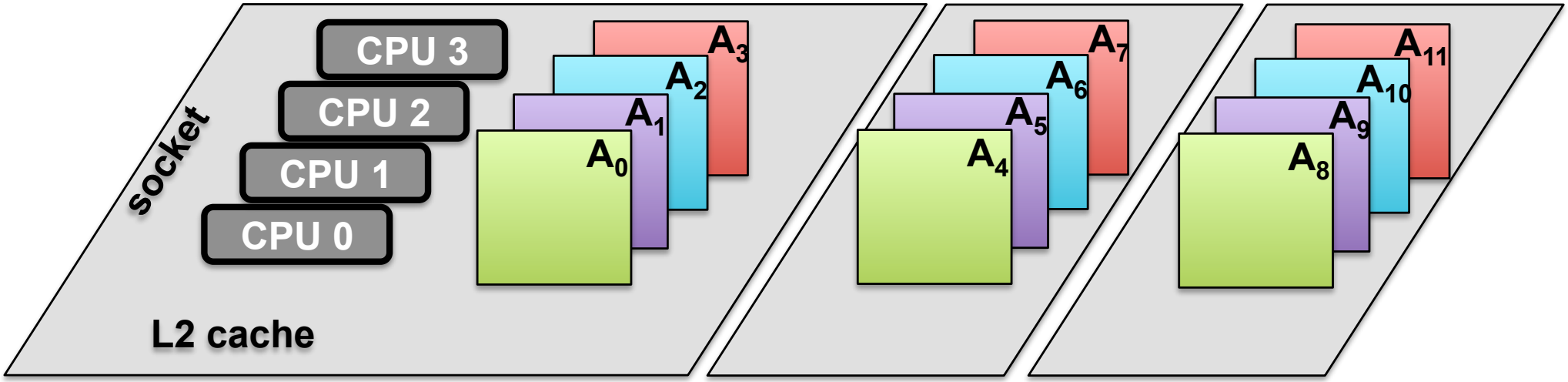
MAGMA Batched Computations CPU

2. Batched computation

loop over the matrices and assign a matrix to each core working on it sequentially and independently

- Since matrices are very small, all the n_cores matrices will fit into L2 cache thus we do not increase L2 cache misses while performing in parallel n_cores computations reaching the best of each core

```
for (i=cpu_id; i<batchount; i+=n_cpu)
```



MAGMA Batched Computations

We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

- There is a **lack of HPC linear algebra software for small problems** especially for GPU
- **CPU**: this can be done easily using existing software infrastructure
- **MIC**: similarly to CPU but, however, today it requires optimizing BLAS routines

MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:

- Many parameter need to be considered, such as:
  - Number of registers, cache size, vectorization AVX2, prefectching, intrinsic versus simple loop, etc…
  - Data Access Optimizations and Loop Transformation Techniques
  - Register Data Reuse and Locality
  - Effect of the Multi-threading
  - Effect of the NUMA-socket and Memory Location

Simply

Need performance analysis (theoretical and tools)

Many code designs, and each may need tuning

MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:

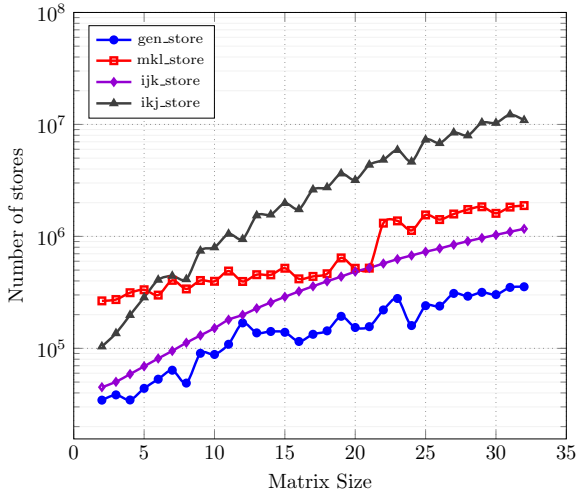
```
void matMulBlk_v3(int M, int N, int K,
                  double alpha, const double* restrict A, int lda,
                                const double* restrict B, int ldb,
                  double beta, double* C, int ldc,
                  int localtid,int threadspergroup)
{

    int x,y,i,j,k,l;
    int Ecol,col,row;

    //local matrix pointers specific to thread
    const double *myA,*myB,*aa,*bb;
    double *myC,*cc;
    int rowstart,colstart,rowstride,colstride;


    __m512d va;
    __m512d vb0, vb1, vb2, vb3, vb4, vb5, vb6, vb7;
    __m512d vc;
    __m512d vy0, vy1, vy2, vy3, vy4, vy5, vy6, vy7;


    rowstart = localtid % 2;
    colstart = localtid / 2;
    rowstride   = threadspergroup==1 ? 1 : 2;
    colstride   = threadspergroup==1 ? 1 : threadspergroup/2;

    for(Ecol=colstart*BLK_N;Ecol<N;Ecol+=BLK_N*colstride)
    {
        myB=B+ldb*Ecol;
        myC=C+ldc*Ecol;

        for(row=rowstart*BLK_M;row<M;row+=BLK_M*rowstride)
        {
            myA=A+row;
            cc=myC+row;

            #pragma unroll
            for(int i=0; i < BLK_K;++i)
                _mm_prefetch((const char *)&myA[(i)*lda], _MM_HINT_T0);
            #pragma unroll
            for(int i=0; i < BLK_N;++i)
                _mm_prefetch((const char *)&myB[(i)*ldb], _MM_HINT_T0);



            //initialize the 8x8 BLK of product Matrix VY
            vy0=_mm512_set1_pd (0.0);
            vy1=_mm512_set1_pd (0.0);
            vy2=_mm512_set1_pd (0.0);
            vy3=_mm512_set1_pd (0.0);
            vy4=_mm512_set1_pd (0.0);
            vy5=_mm512_set1_pd (0.0);
            vy6=_mm512_set1_pd (0.0);
            vy7=_mm512_set1_pd (0.0);


            #pragma unroll
            for(col=0;col<K;col+=BLK_K)
            {
                aa  = myA+lda*col;
                bb  = myB+col;
                // computing va(:,0)*vb(0,BLK_N) of vy(:,BLK_N)
                va  = _mm512_extload_pd(aa+0*lda,_MM_UPCONV_PD_NONE,_MM_BROADCAST64_NONE,_MM_HINT_T0);
                vb0  = _mm512_extload_pd(bb+0+0*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy0 = _mm512_fmadd_pd (va,vb0,vy0);
                vb1  = _mm512_extload_pd(bb+0+1*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy1 = _mm512_fmadd_pd (va,vb1,vy1);
                vb2  = _mm512_extload_pd(bb+0+2*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy2 = _mm512_fmadd_pd (va,vb2,vy2);
                vb3  = _mm512_extload_pd(bb+0+3*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy3 = _mm512_fmadd_pd (va,vb3,vy3);
                vb4  = _mm512_extload_pd(bb+0+4*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy4 = _mm512_fmadd_pd (va,vb4,vy4);
                vb5  = _mm512_extload_pd(bb+0+5*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy5 = _mm512_fmadd_pd (va,vb5,vy5);
                vb6  = _mm512_extload_pd(bb+0+6*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy6 = _mm512_fmadd_pd (va,vb6,vy6);
                vb7  = _mm512_extload_pd(bb+0+7*ldb,_MM_UPCONV_PD_NONE,_MM_BROADCAST_1X8,_MM_HINT_NONE);
                vy7 = _mm512_fmadd_pd (va,vb7,vy7);
```

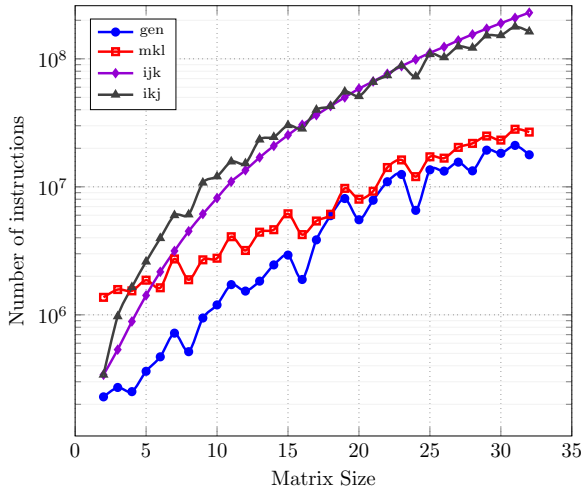
```
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
echo 'void matMulBlk_v3(int M, int N, int K,'                              >> $outputfname
echo '                  double alpha, const double* restrict A, int lda,'  >> $outputfname
echo '                                const double* restrict B, int ldb,'  >> $outputfname
echo '                  double beta, double* C, int ldc,'                  >> $outputfname
echo '                  int localtid,int threadspergroup)'                 >> $outputfname
echo '{'                                                                   >> $outputfname
echo ''                                                                    >> $outputfname
echo '    int x,y,i,j,k,l; '                                              >> $outputfname
echo '    int Ecol,col,row; '                                              >> $outputfname
echo ' '                                                                   >> $outputfname
echo '    //local matrix pointers specific to thread '                     >> $outputfname
echo '    const double *myA,*myB,*aa,*bb; '                                >> $outputfname
echo '    double *myC,*cc; '                                               >> $outputfname
echo '    int rowstart,colstart,rowstride,colstride;'                      >> $outputfname
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
echo '    __m512d va; '                                                    >> $outputfname
echo '    __m512d vb; '                                                    >> $outputfname
#echo '     __m512d' $vbreg                                                >> $outputfname
echo '    __m512d vc; '                                                    >> $outputfname
echo '    __m512d' $vyreg                                                  >> $outputfname
```

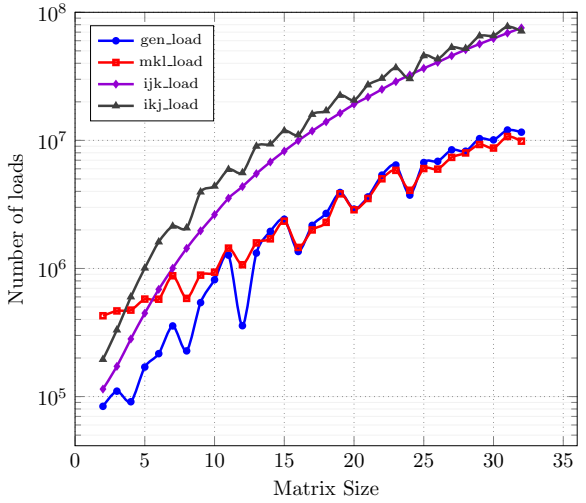
```
#*****************************************************************************************
# start the main code
#*****************************************************************************************
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
echo '    rowstart = localtid % 2;'                                        >> $outputfname
echo '    colstart = localtid / 2;'                                        >> $outputfname
echo '    rowstride   = threadspergroup==1 ? 1 : 2;'                       >> $outputfname
echo '    colstride   = threadspergroup==1 ? 1 : threadspergroup/2;'       >> $outputfname
echo ''                                                                    >> $outputfname
echo '    for(Ecol=colstart*BLK_N;Ecol<N;Ecol+=BLK_N*colstride)'           >> $outputfname
echo '    {'                                                               >> $outputfname
echo '        myB=B+ldb*Ecol;'                                             >> $outputfname
echo '        myC=C+ldc*Ecol;'                                             >> $outputfname
echo ''                                                                    >> $outputfname
echo '        for(row=rowstart*BLK_M;row<M;row+=BLK_M*rowstride)'          >> $outputfname
echo '        {'                                                           >> $outputfname
echo '            myA=A+row;'                                              >> $outputfname
echo '            cc=myC+row;'                                             >> $outputfname
echo ''                                                                    >> $outputfname
echo '            #pragma unroll'                                          >> $outputfname
echo '            for(int i=0; i < BLK_K;++i)'                             >> $outputfname
echo '                _mm_prefetch((const char *)&myA[(i)*lda], _MM_HINT_T0);' >> $outputfname
echo '            #pragma unroll'                                          >> $outputfname
echo '            for(int i=0; i < BLK_N;++i)'                             >> $outputfname
echo '                _mm_prefetch((const char *)&myB[(i)*ldb], _MM_HINT_T0);' >> $outputfname
echo ''                                                                    >> $outputfname
echo ''                                                                    >> $outputfname
```

MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:

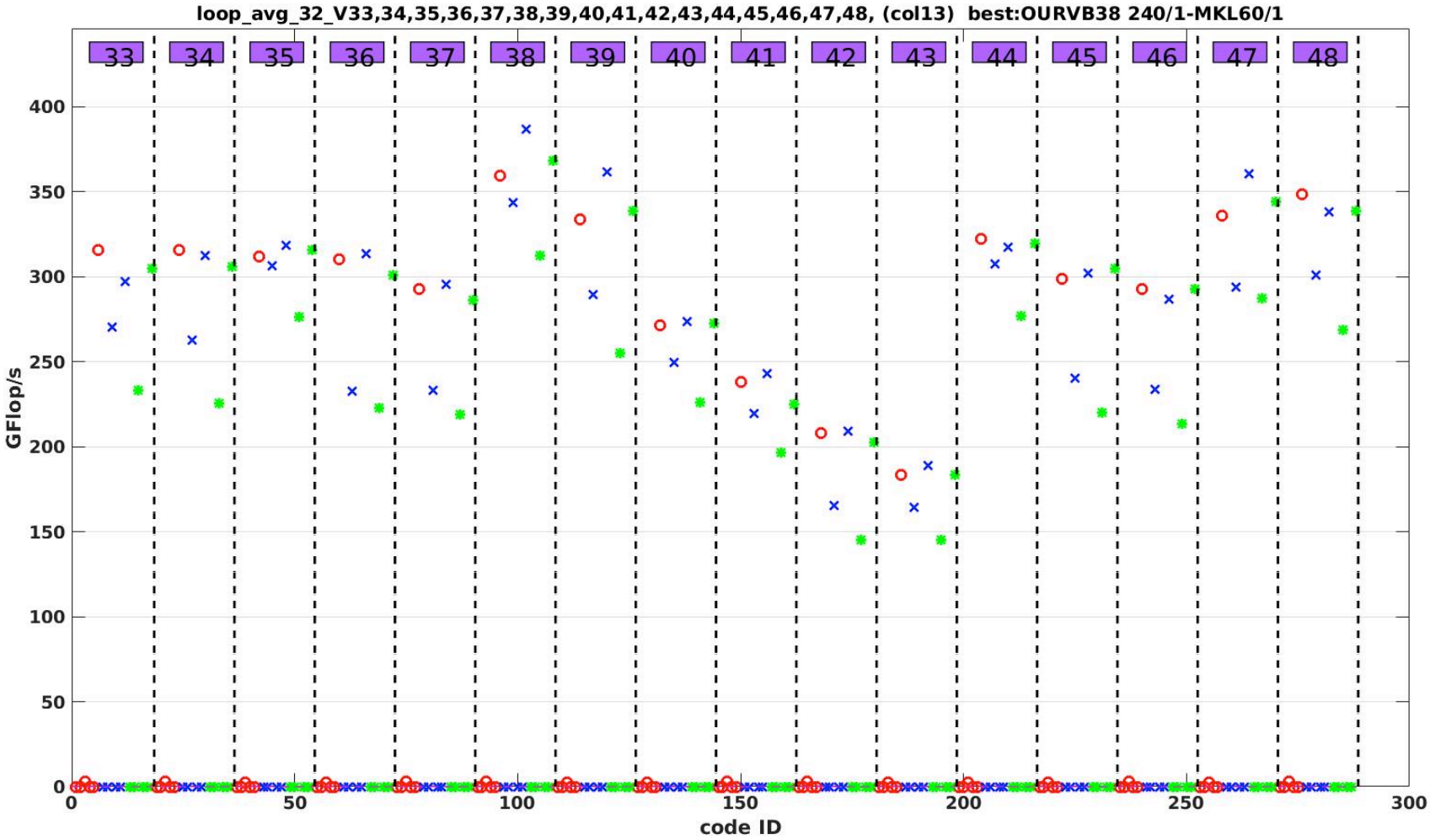






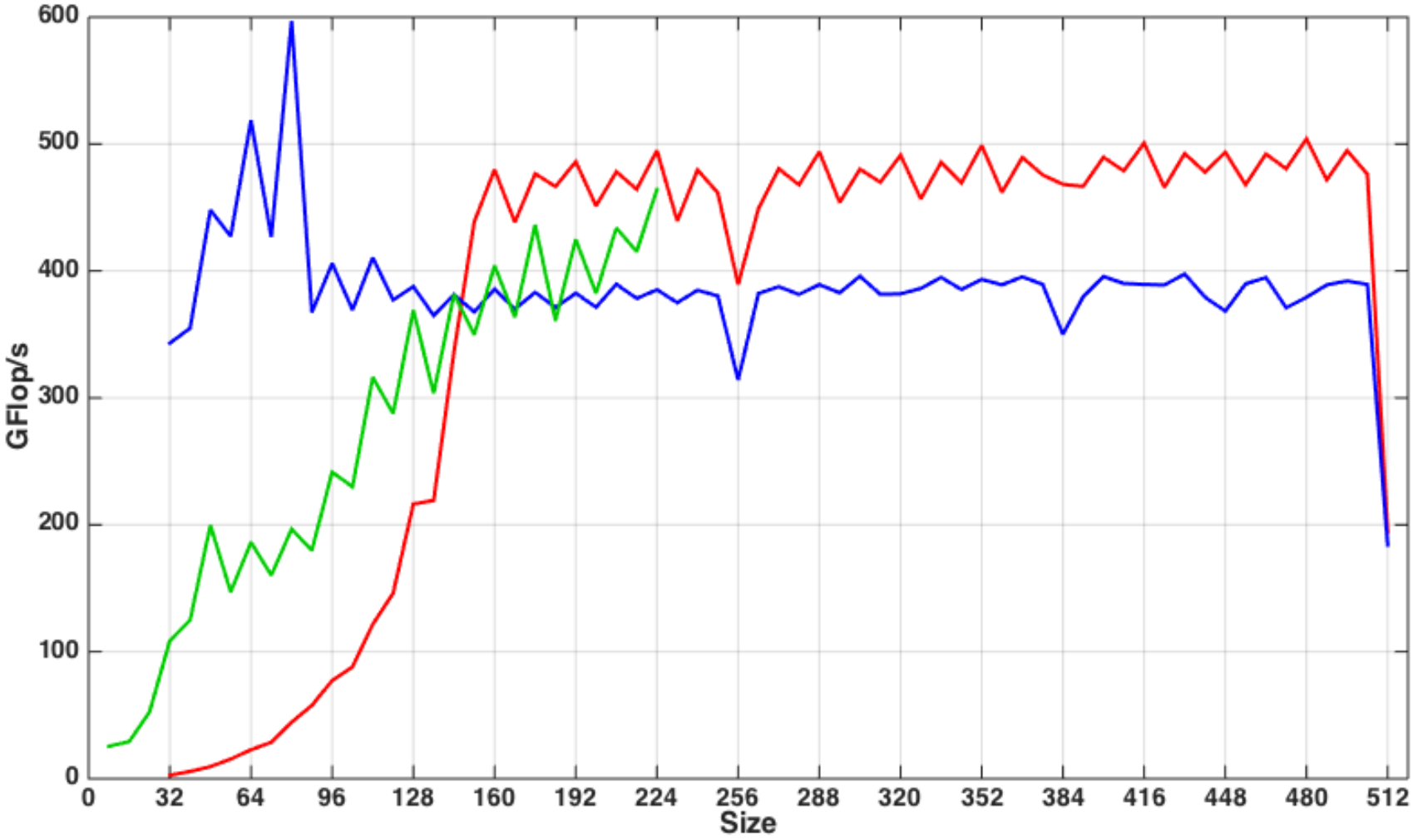
MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:



MAGMA Batched Computations MIC

Programming model: auto-generation and auto-tuning:



MAGMA Batched Computations

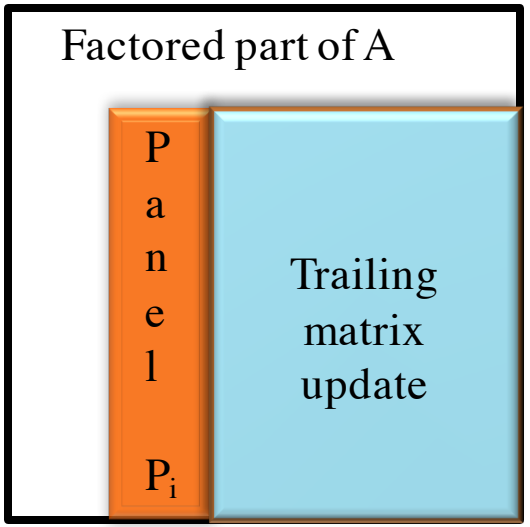
We present here a feasibility design study, the idea is to target the new high-end technologies.

Key observations and current situation:

- There is a **lack of HPC linear algebra software for small problems** especially for GPU
- **CPU**: this can be done easily using existing software infrastructure
- **MIC**: Similarly to CPU but, today it requires optimizing BLAS routines
- **GPU**: are efficient for large data parallel computations, and therefore have often been used in combination with CPUs, where the CPU handles the memory bound and difficult tasks to be parallelized while the GPU is used for data intensive tasks
- **What programming model is best for small problems?**

Algorithmic basics

- **Linear solver *Ax=b*** follow the LAPACK-style algorithmic design
- Two distinctive phases
  - panel factorization: latency-bound workload
  - trailing matrix update: compute-bound operation



References:

1. **A. Haidar, S. Tomov, P. Luszczek, and J. Dongarra.**
   MAGMA Embedded: Towards a Dense Linear Algebra Library for Energy Efficient Extreme Computing
   IEEE High Performance Extreme Computing Conference IEEE-HPEC 2015, Waltham, MA USA.
   Best paper award

2. **A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra.**
   A Framework for Batched and GPU-resident Factorization Algorithms Applied to Block Householder Transformations
   International Supercomputing Conference IEEE-ISC 2015, Frankfurt, Germany.

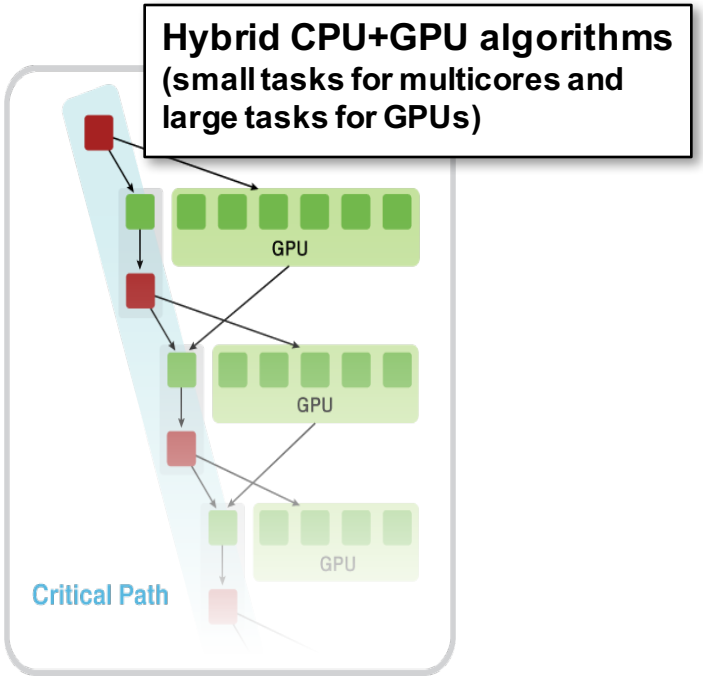
   K. Kabir, A. Haidar, S. Tomov, and J. Dongarra
   On the Design, Development and Analysis of Optimized Matrix-Vector Multiplication Routines for coprocessors.
   International SuperComputing Conference IEEE-ISC 2015, Frankfurt, Germany

3. **A. Haidar, T. Dong, P. Luszczek, S. Tomov and J. Dongarra.**
   Batched Matrix Computations on Hardware Accelerators Based on GPUs.
   International Journal of High Performance Computing Applications 2014.

MAGMA Batched Computations GPU

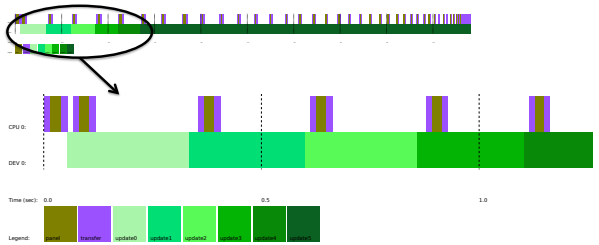
Classical strategies design

- 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**

MAGMA Batched Computations GPU

Classical strategies design

- For large problems performance is driven by the update operations, e.g., Level 3 BLAS (GEMM)

Challenges

- For batched small matrices it is more complicated and requires both phases to be efficient

Proposition

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

MAGMA Batched Computations GPU

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

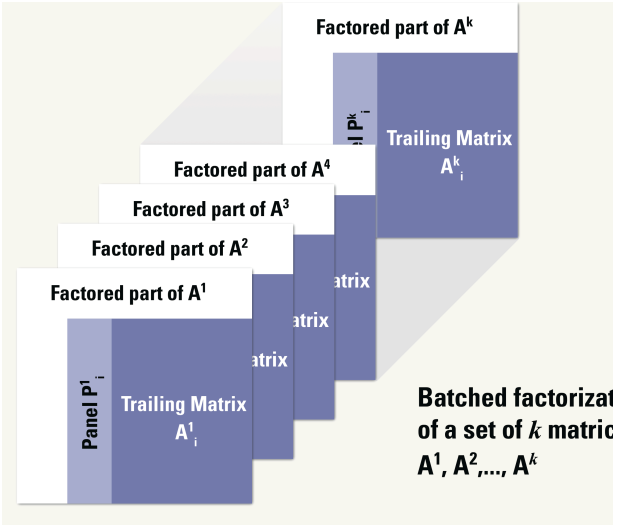
- Our study and experience shows that this procedure provides very good performance for classical GPU kernels but is **not that appealing for batched algorithm** for different reasons.

MAGMA Batched Computations GPU

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)

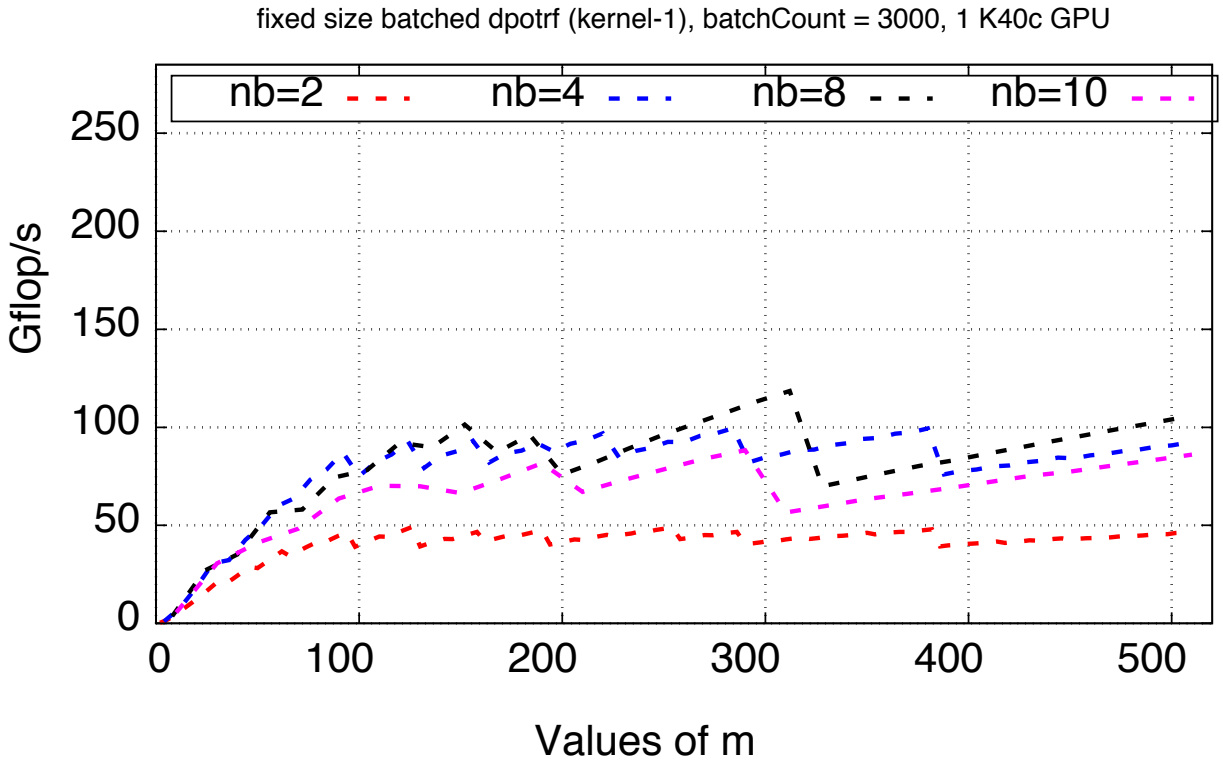
Proposition: custom design per operations type



MAGMA Batched Computations GPU

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.

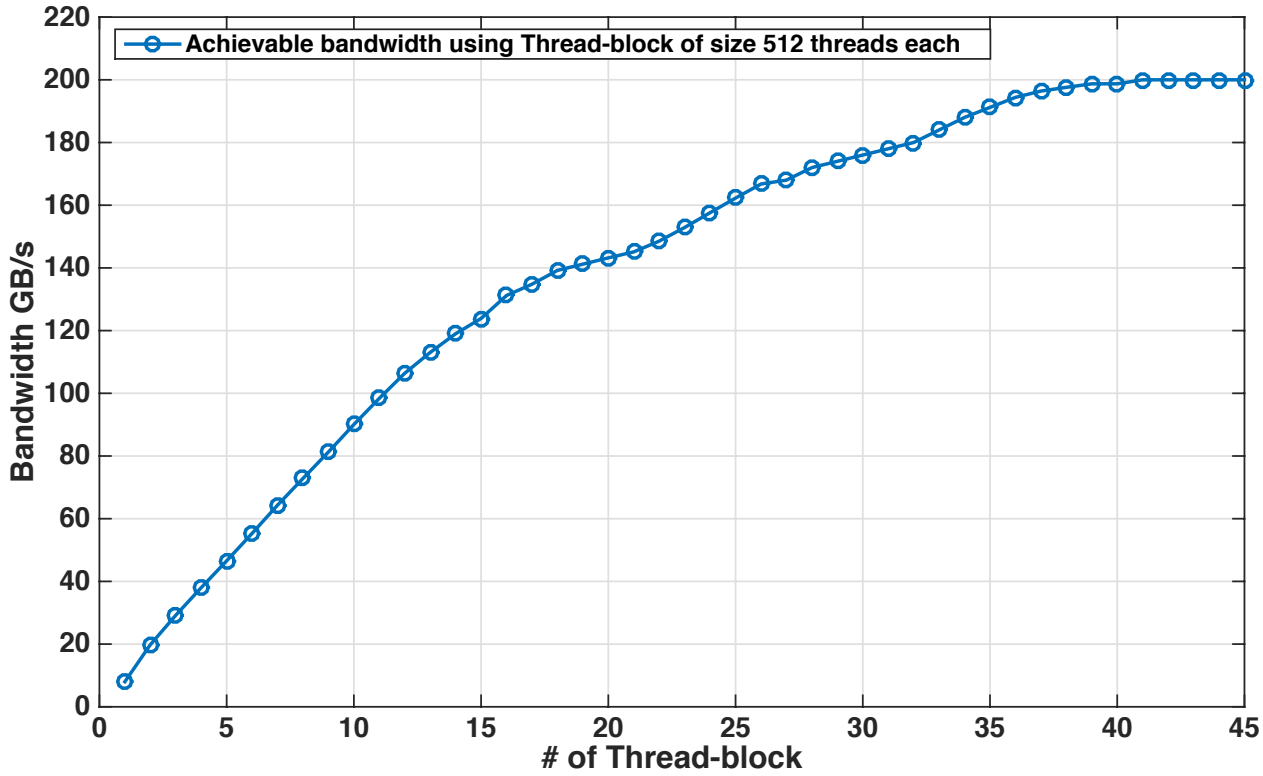


- ✓ **optimized kernel**
- ✓ **using shared memory**
- ✓ **left v.s. right looking**
- ✓ **autotuned**

MAGMA Batched Computations GPU

Performance metrics analysis

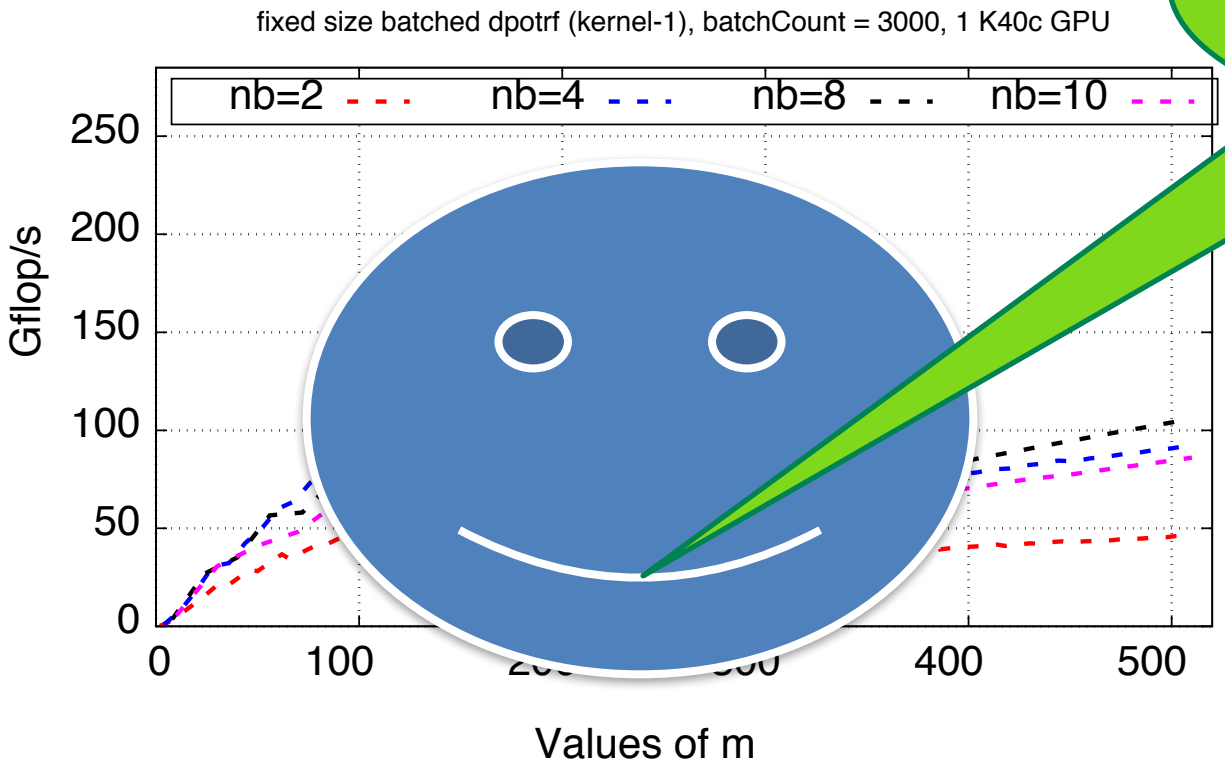
- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory, registers/TB** – load it with data and reuse that data in computations as much as possible.



MAGMA Batched Computations GPU

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.



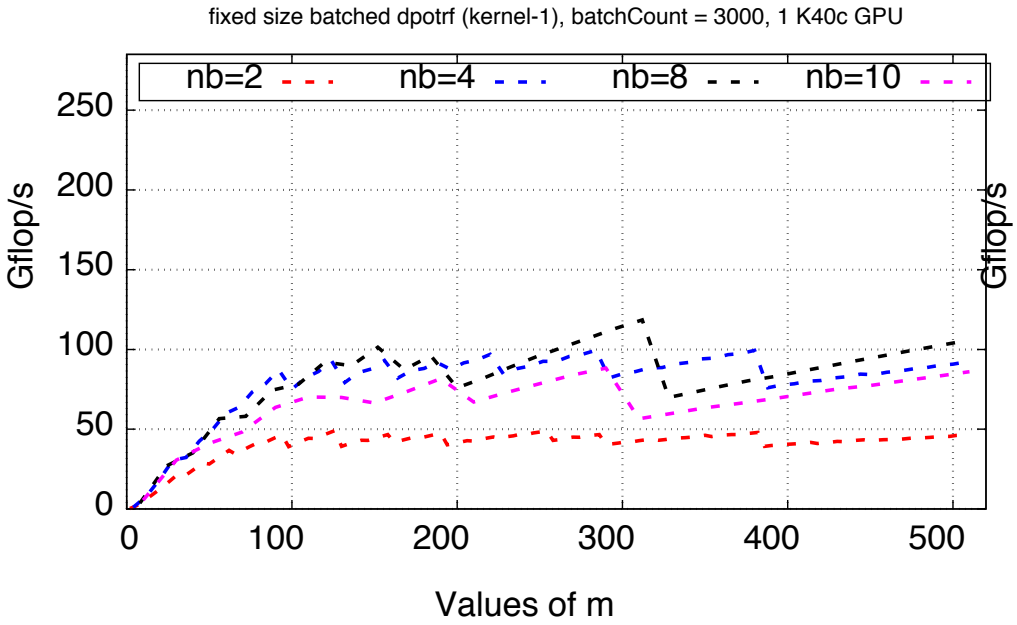
We should focus on the performance analysis and the design of a kernel

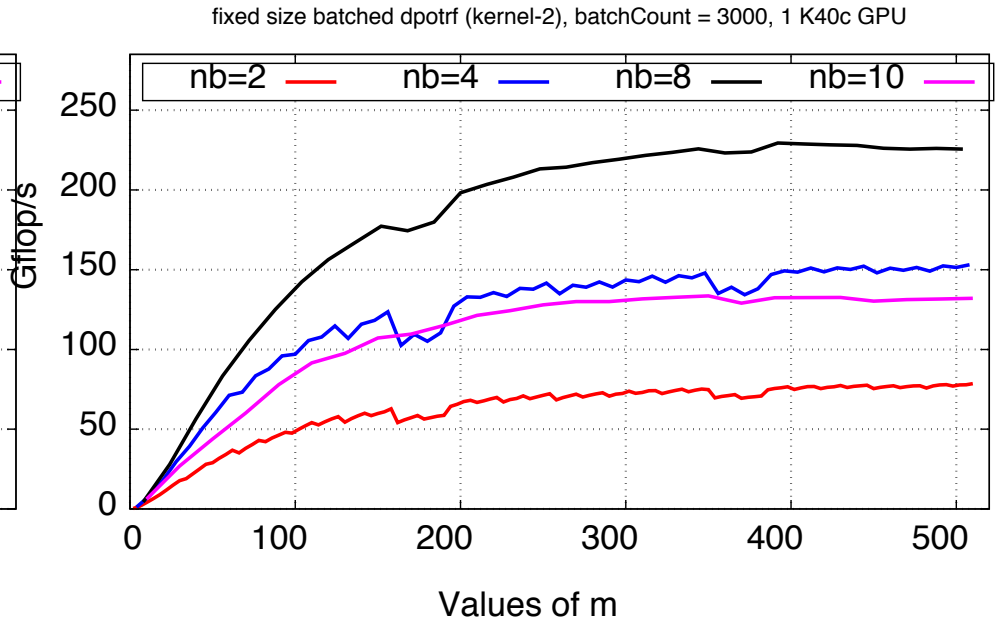
- ✓ **optimized kernel**
- ✓ **using shared memory**
- ✓ **left v.s. right looking**
- ✓ **autotuned**

MAGMA Batched Computations GPU

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.





MAGMA Batched Computations GPU

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

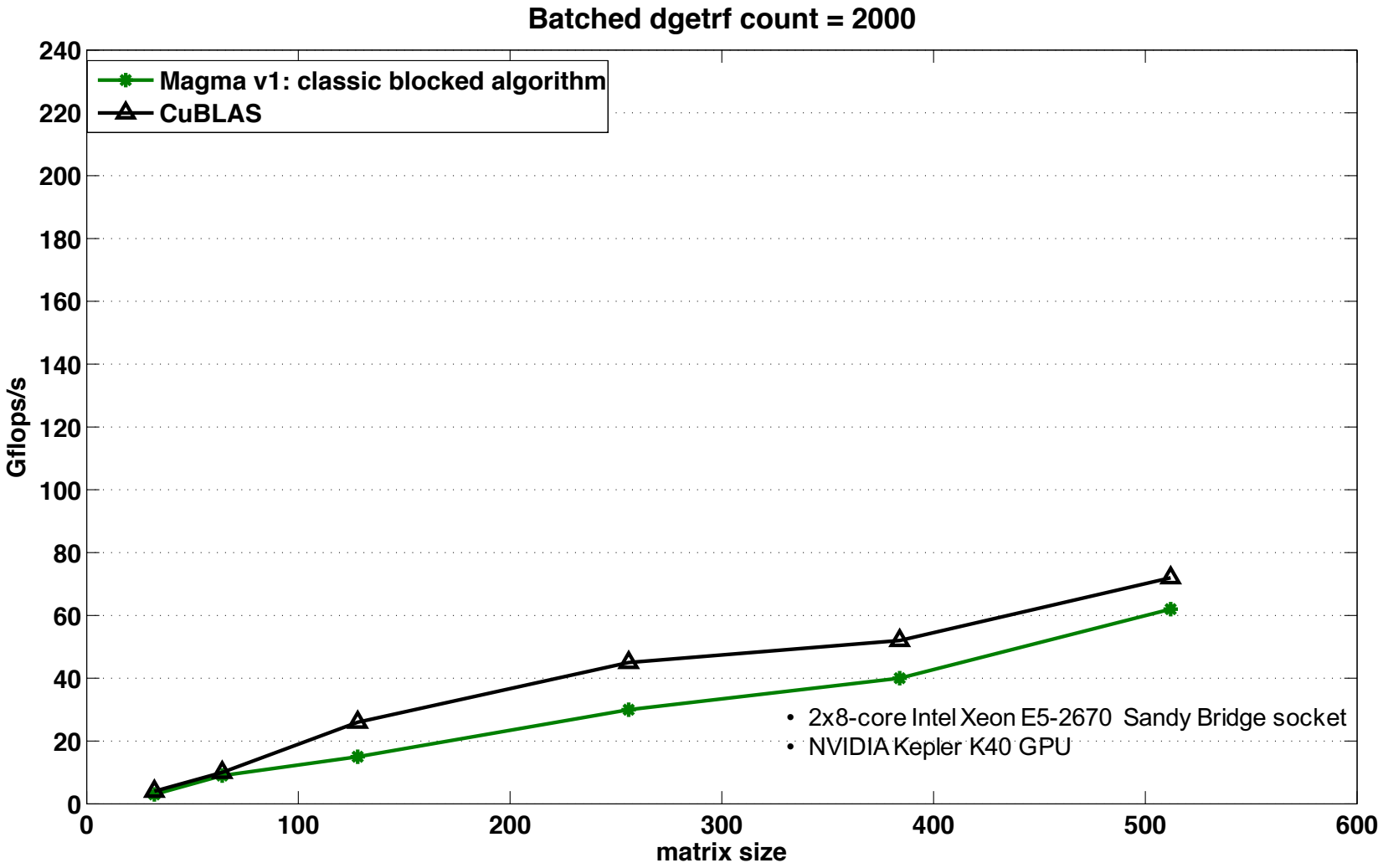
MAGMA Batched Computations GPU

Anatomy of Optimizing an Algorithm:
Performance Analysis and Kernels
Design of the LU Factorization

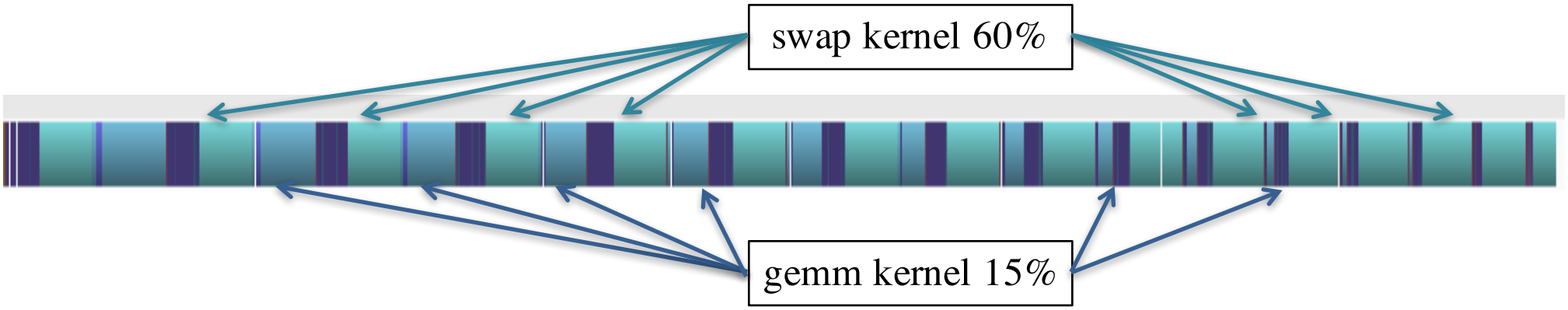
ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations GPU

Consider the LU factorization

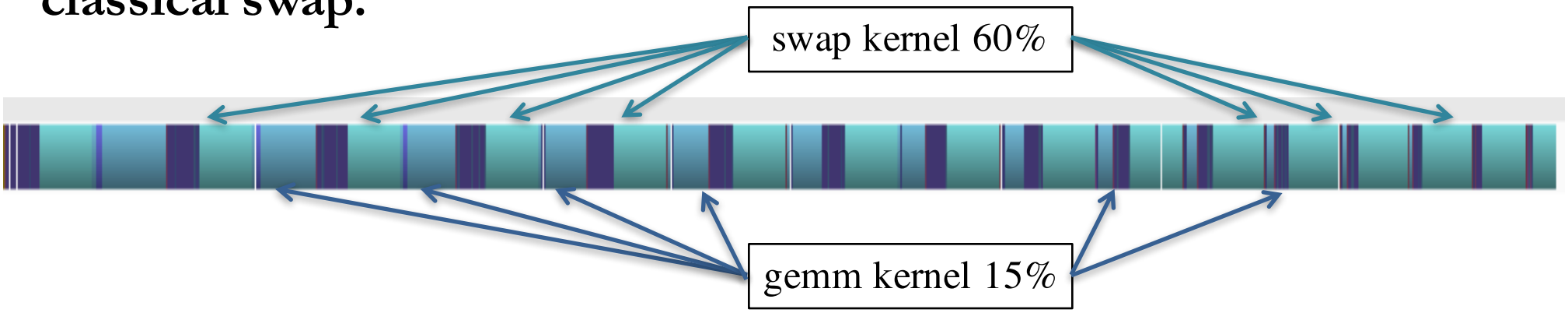


Profile and trace to find bottlenecks

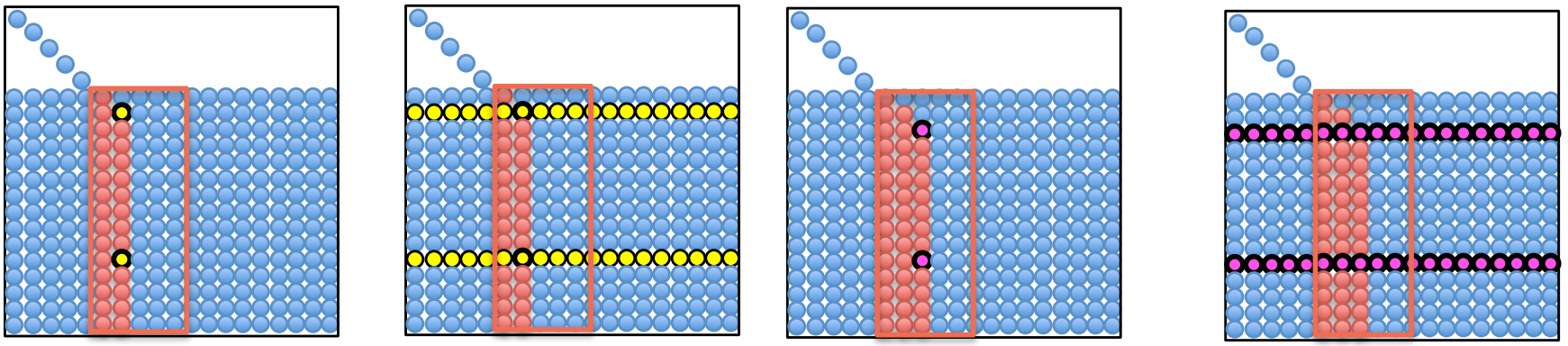


Profile and trace to find bottlenecks

classical swap:

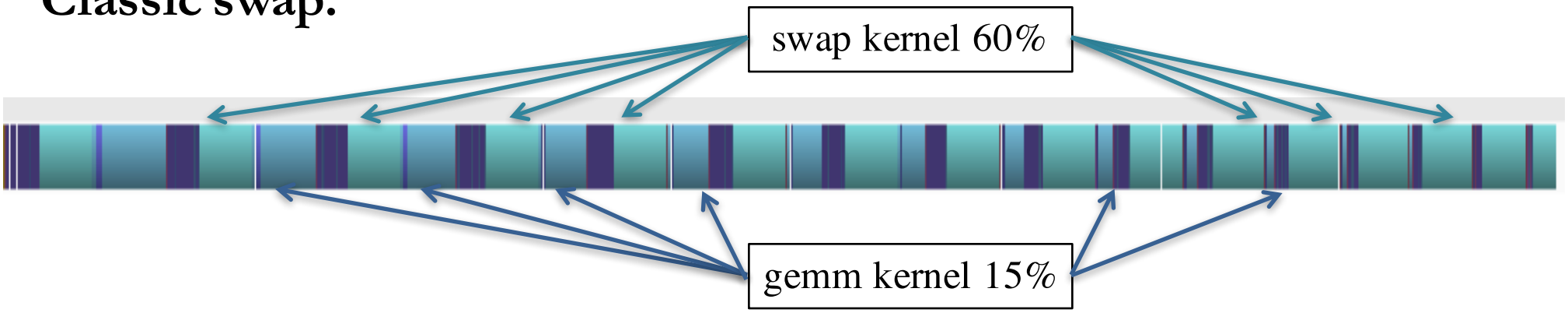


How does the swap work?

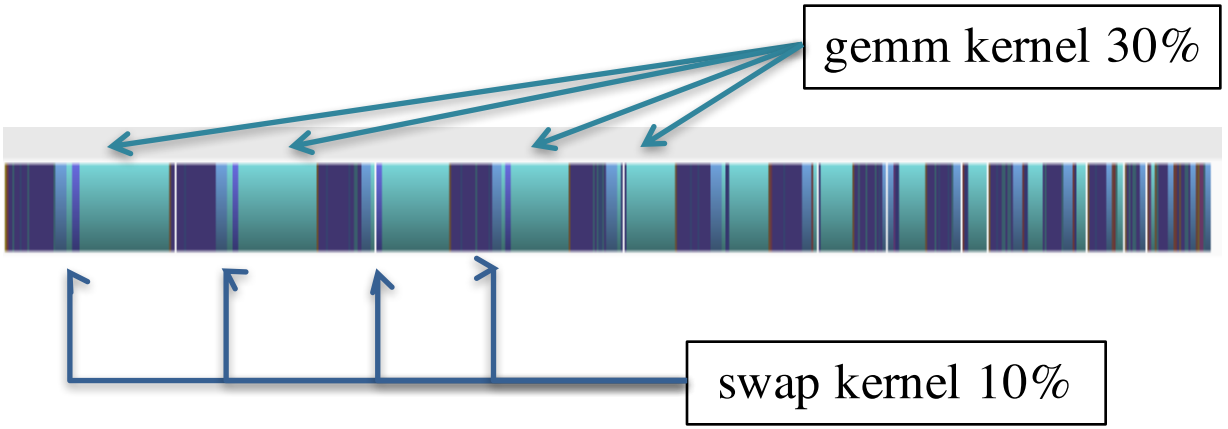


Profile and trace to find bottlenecks

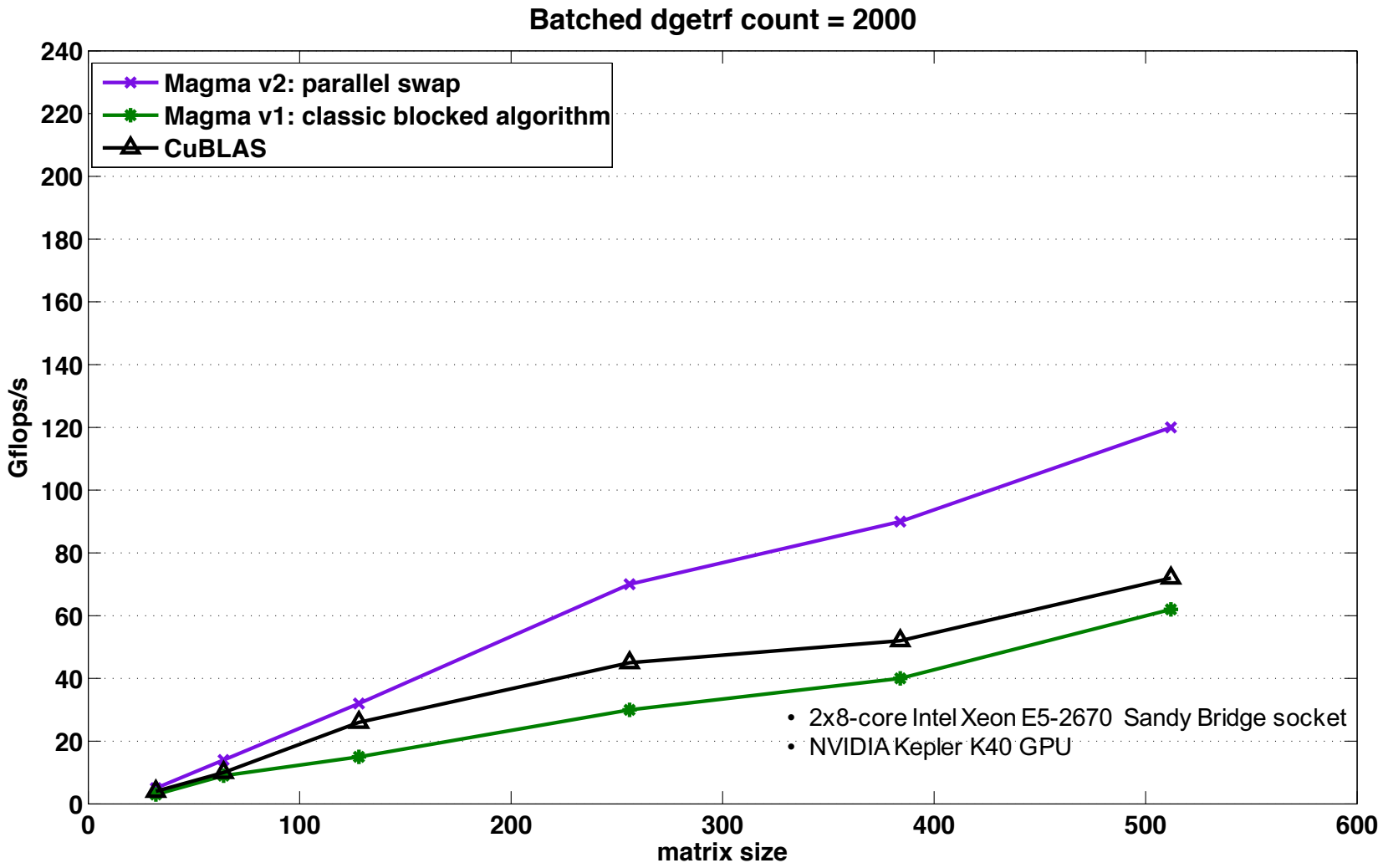
Classic swap:



Parallel swap:

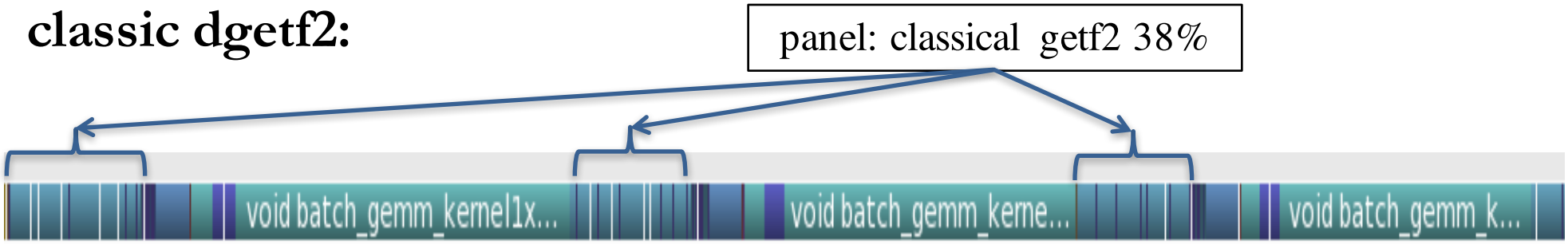


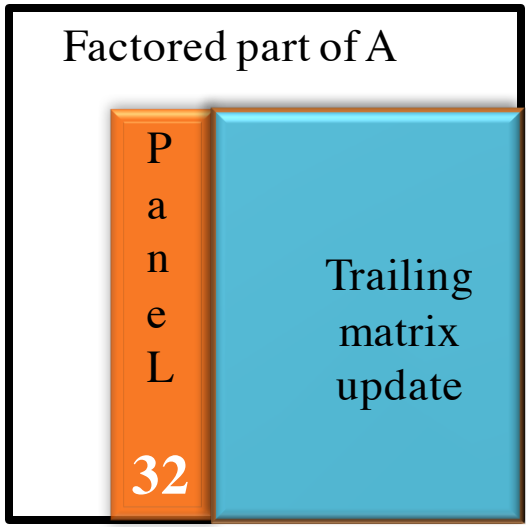
MAGMA Batched Computations GPU



MAGMA Batched Computations GPU

Panel factorization classic dgetf2:





Bottlenecks:

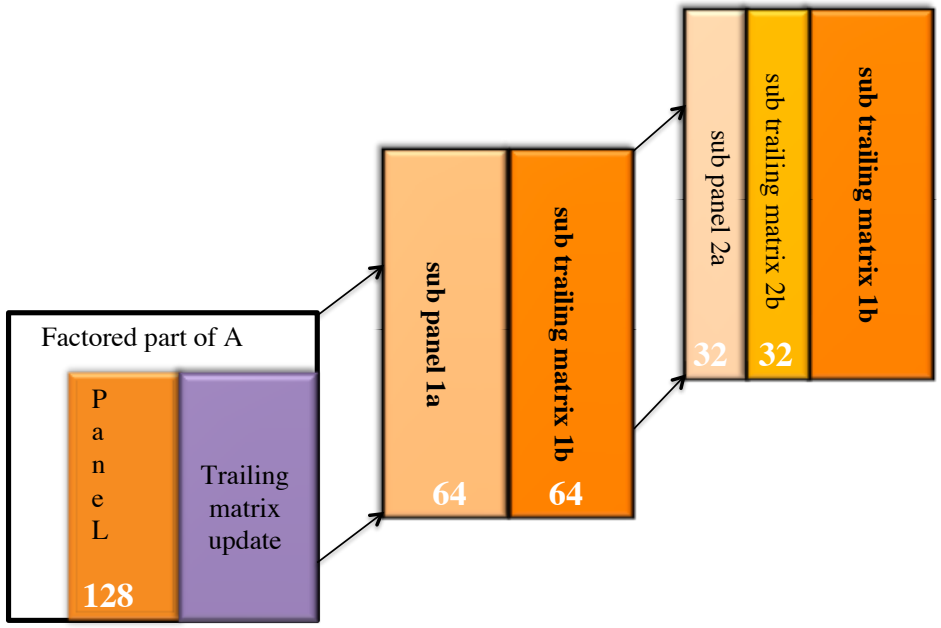
- *nb* large: panel get slower **--> very bad performance.**
- *nb* small: panel get faster but the update is not anymore efficient since dealing with gemm's of small sizes **--> very bad performance**.
- trade-off ? No effect, since we are talking about small size.

Proposition:

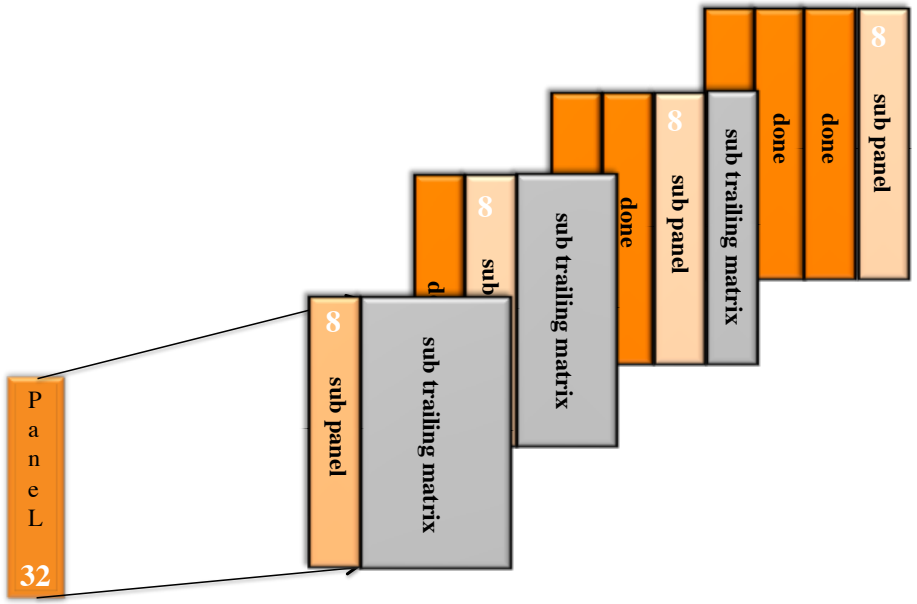
- We propose to develop **two layers blocking**: a recursive and nested blocking technique that block also the panel.

MAGMA Batched Computations GPU

Two-layers blocking:



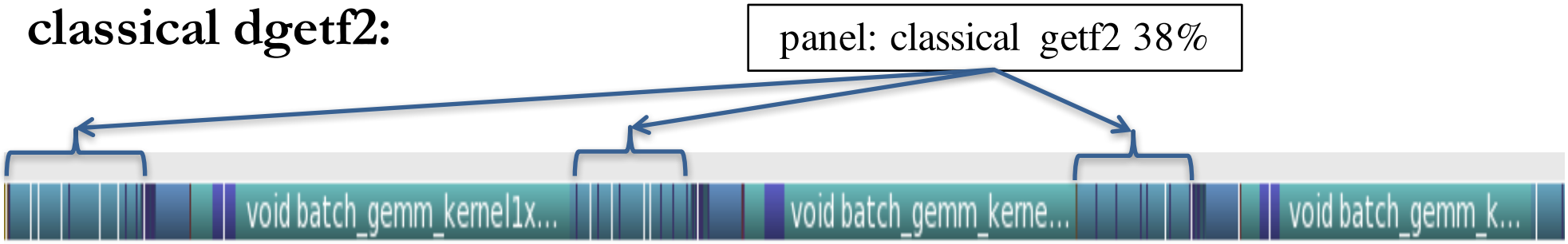
(a) Recursive nested blocking fashion.



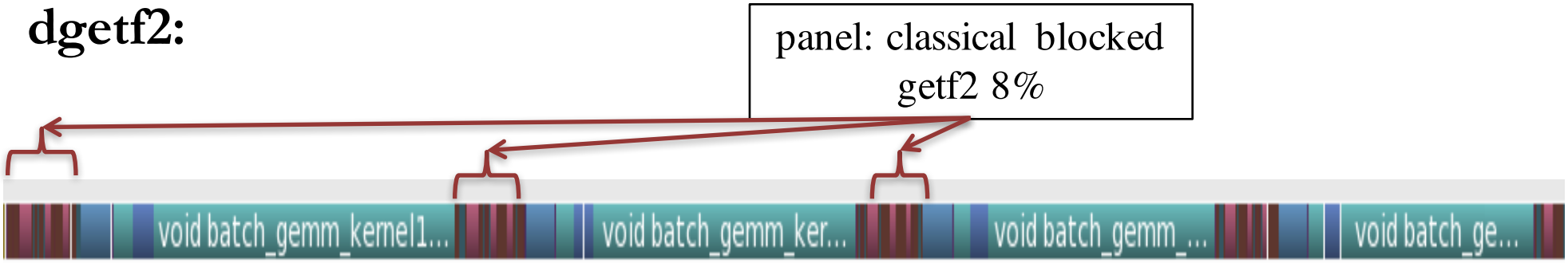
(b) Classical blocking fashion.

MAGMA Batched Computations GPU

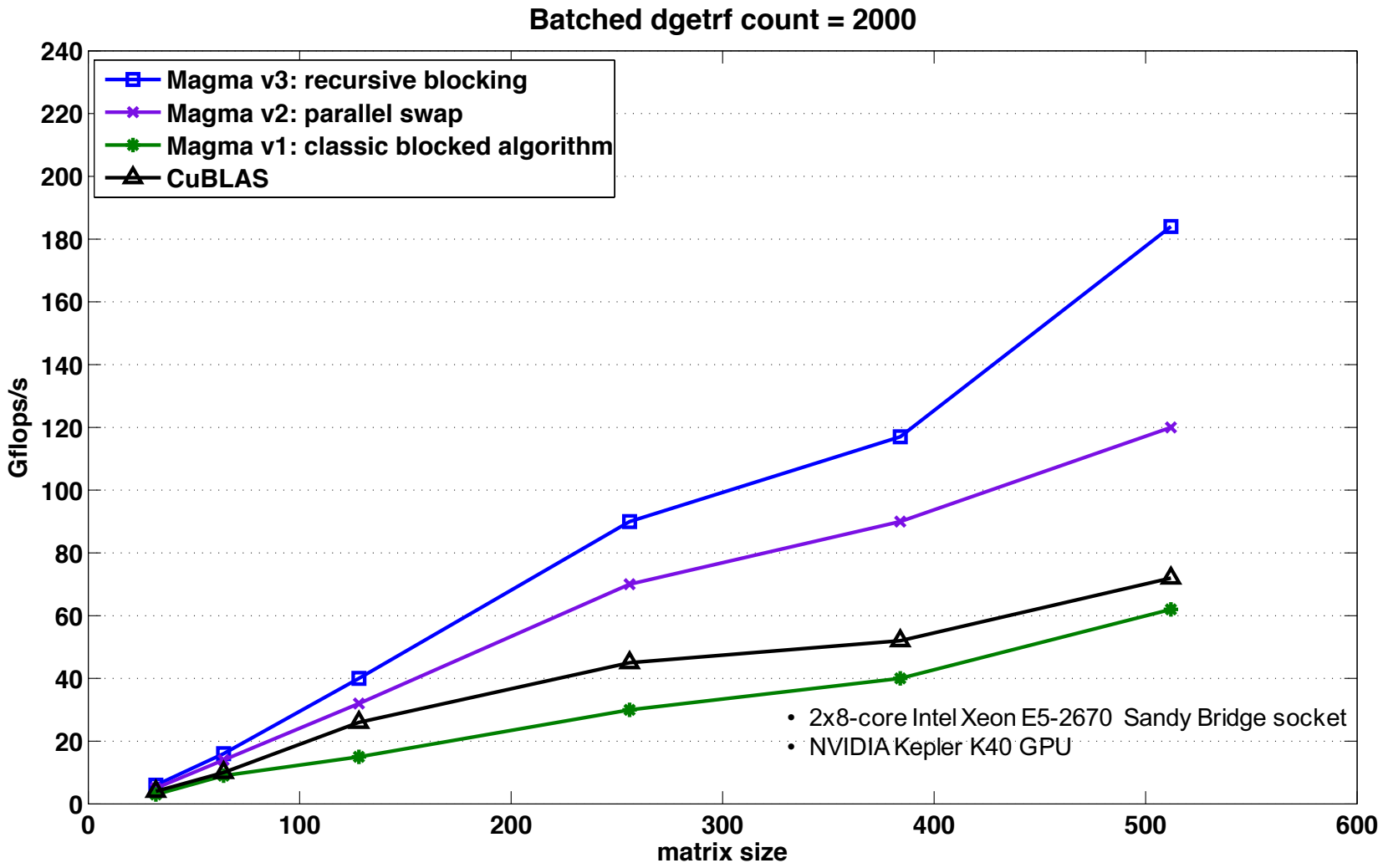
panel factorization
classical dgetf2:



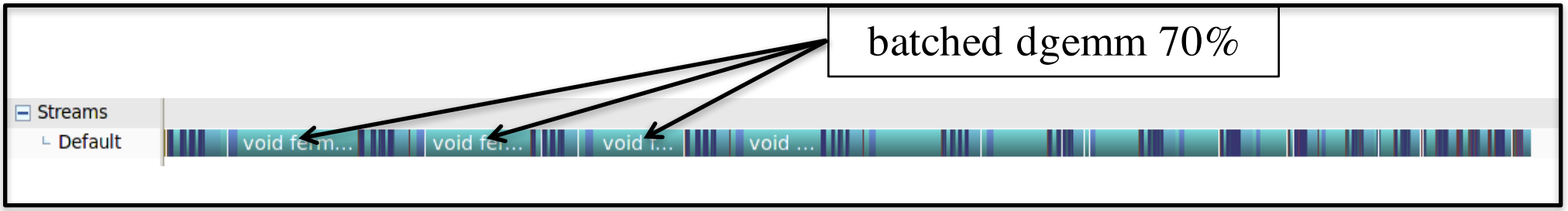
Recursive blocking of
dgetf2:



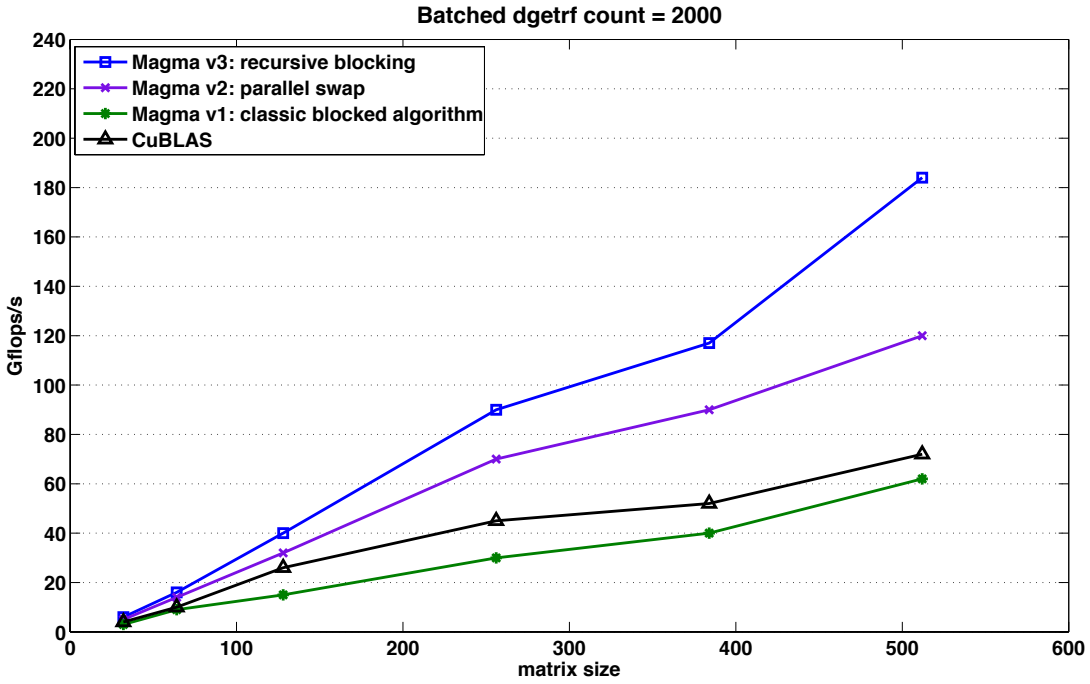
MAGMA Batched Computations GPU



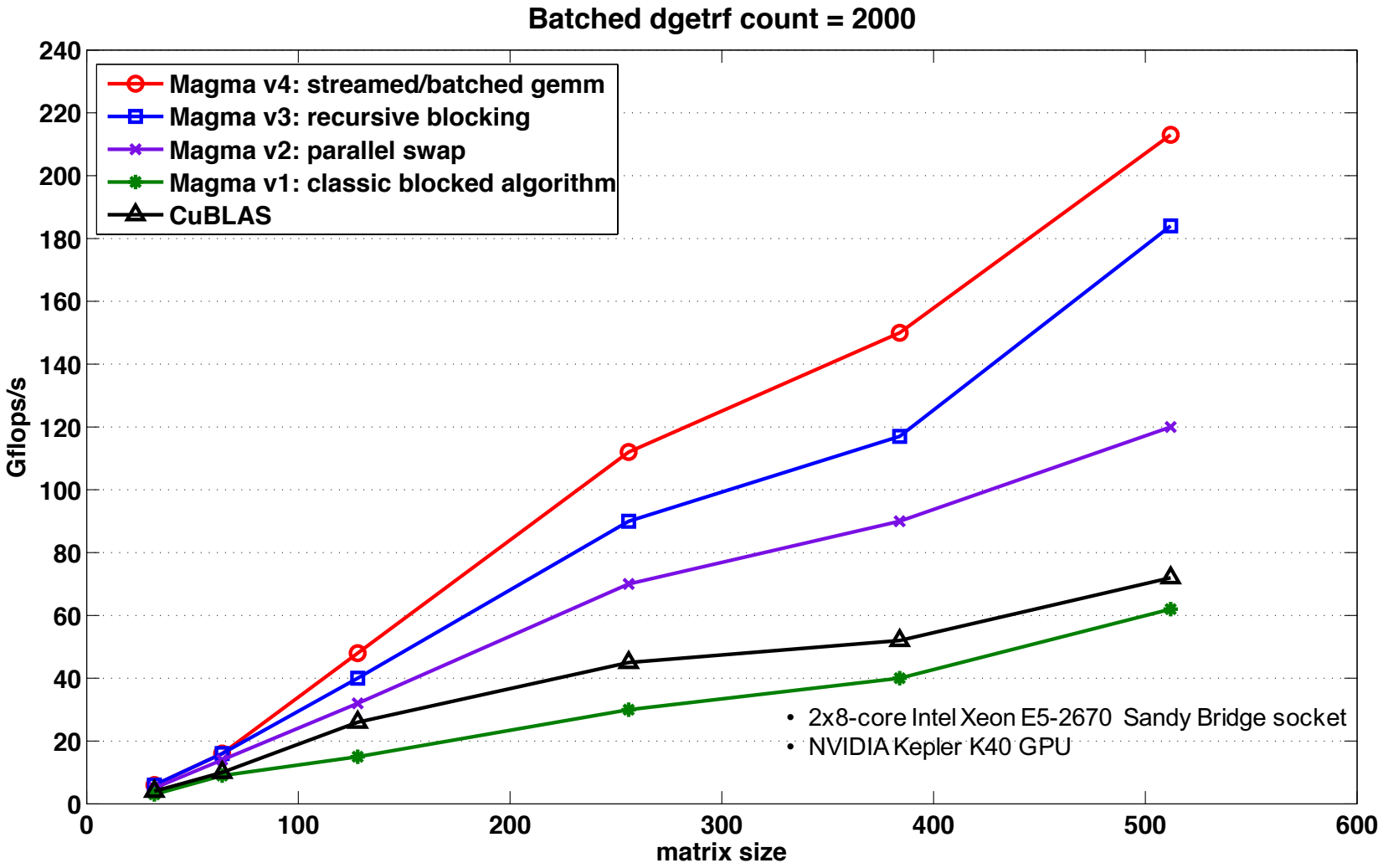
MAGMA Batched Computations GPU



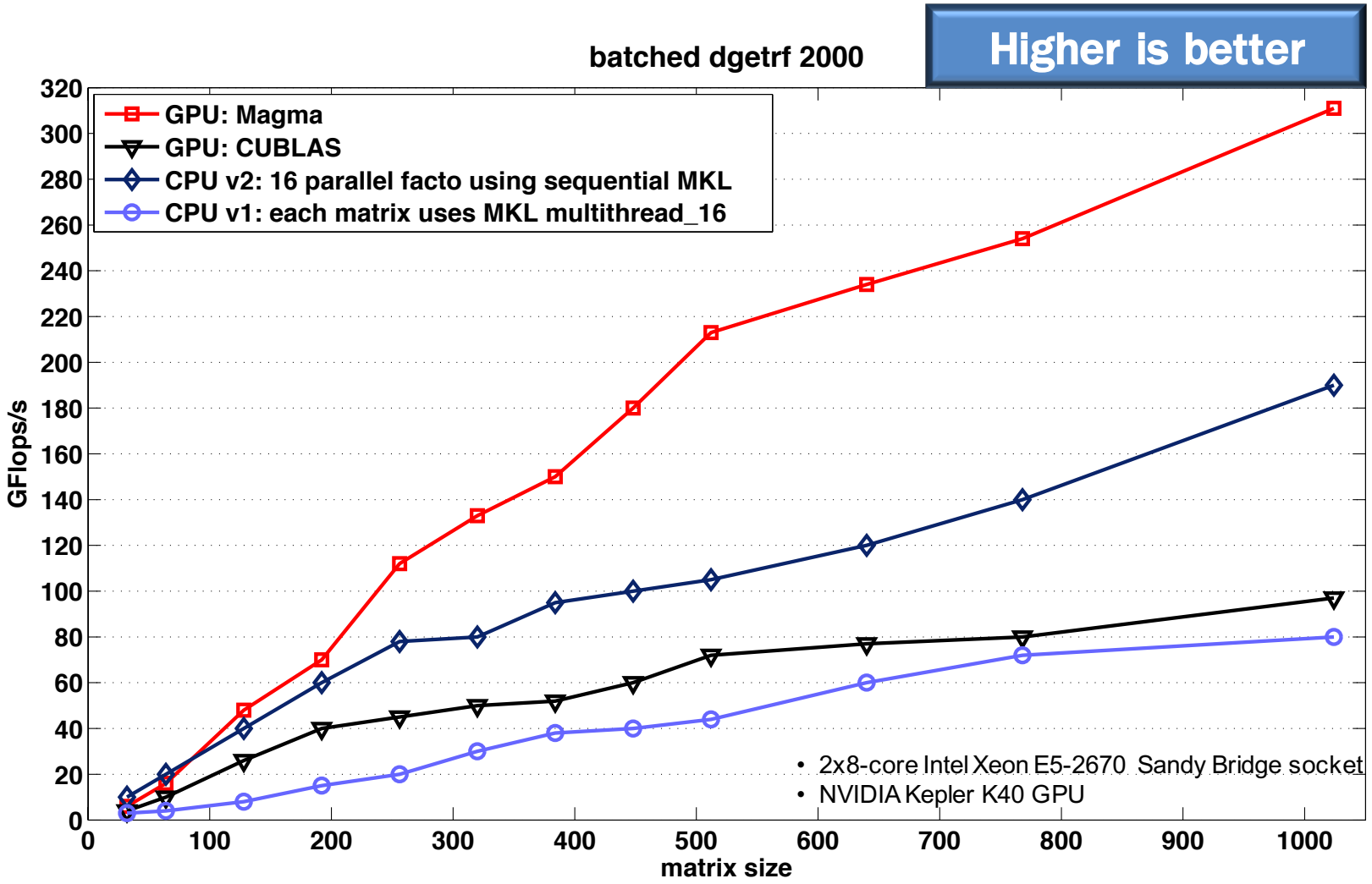
Try to tune and optimize batched dgemm



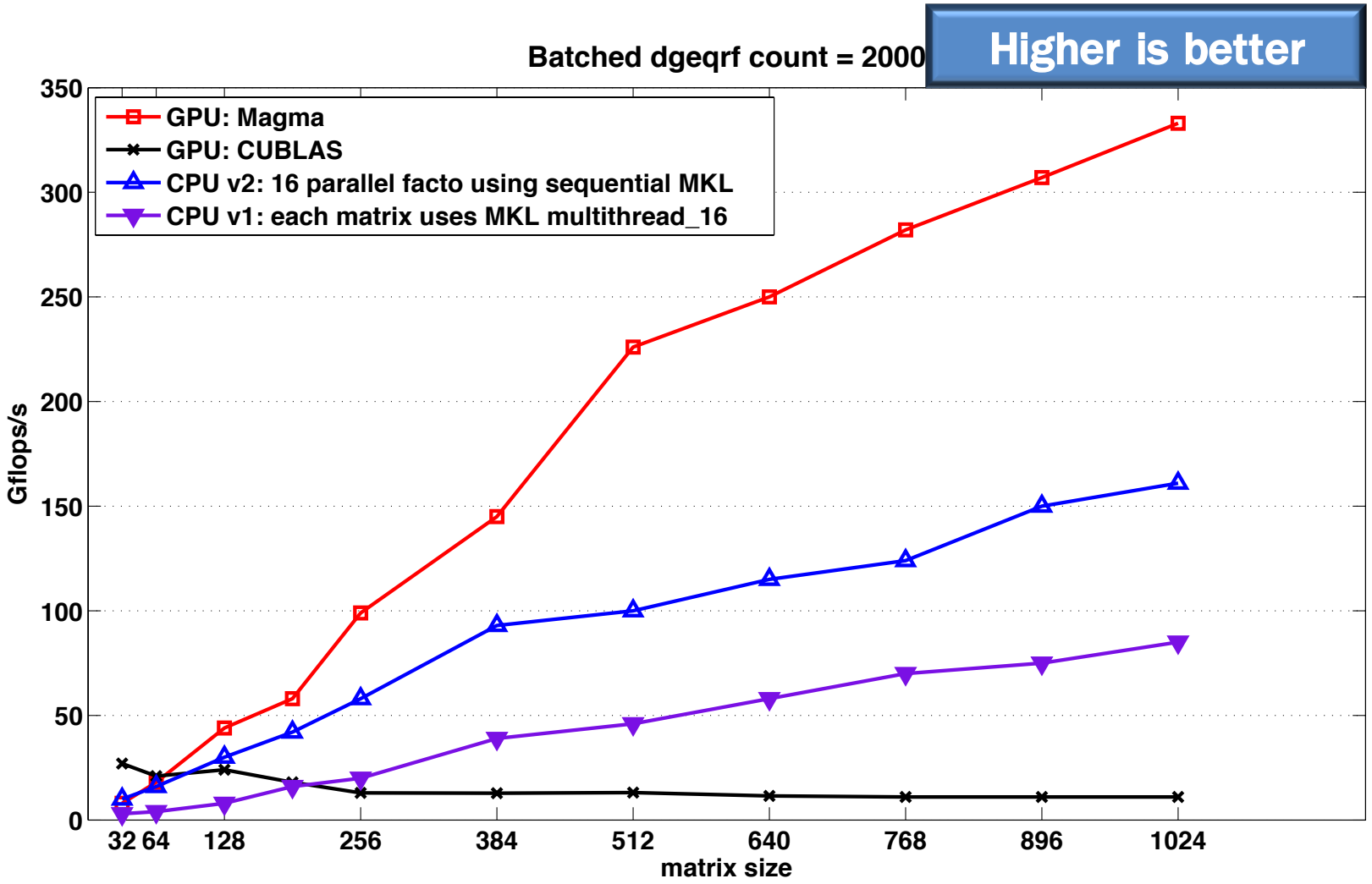
MAGMA Batched Computations GPU



MAGMA Batched Computations Comparison to CPUs



MAGMA Batched Computations Comparison to CPUs

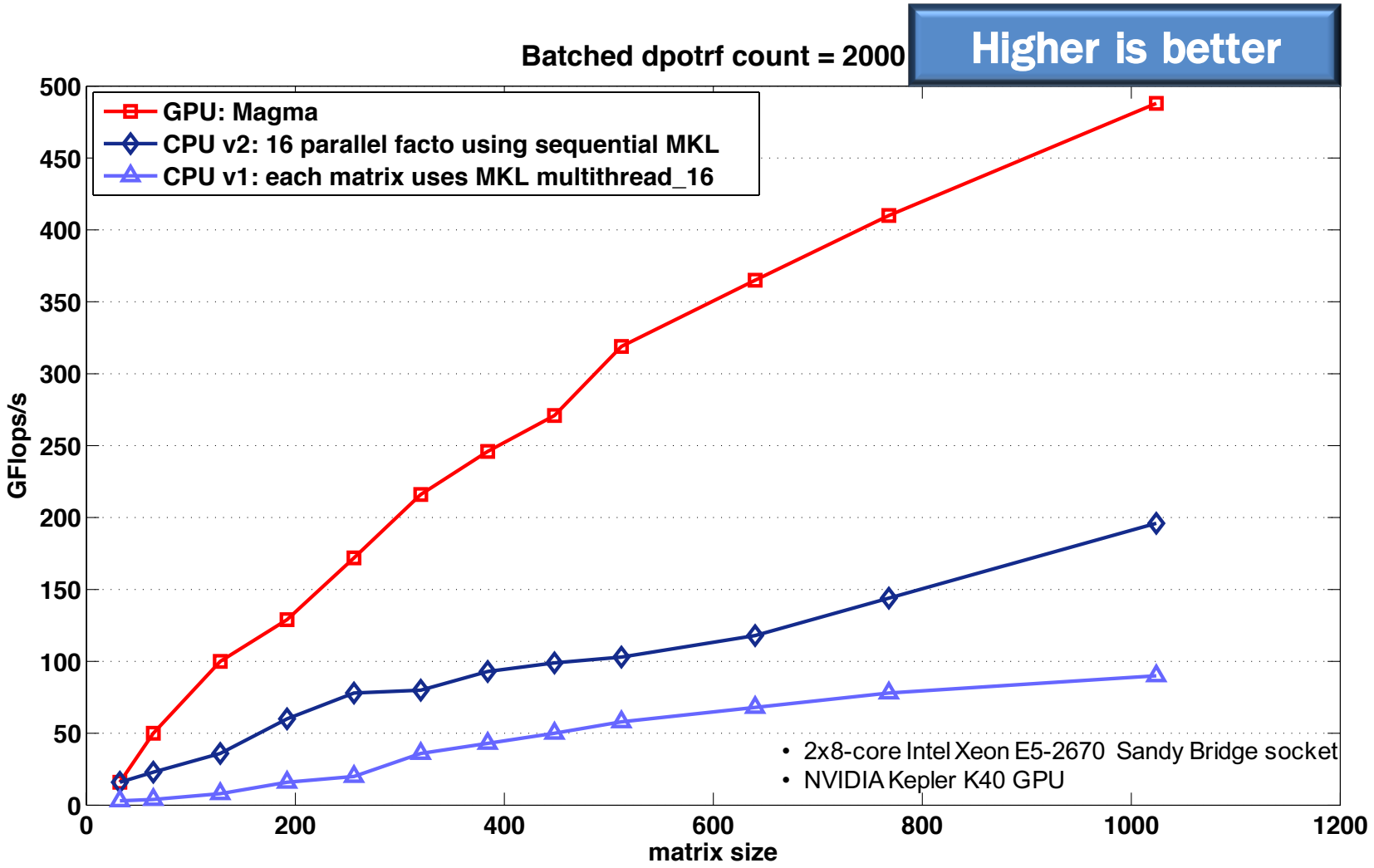


- 2x8-core Intel Xeon E5-2670 Sandy Bridge socket
- NVIDIA Kepler K40 GPU

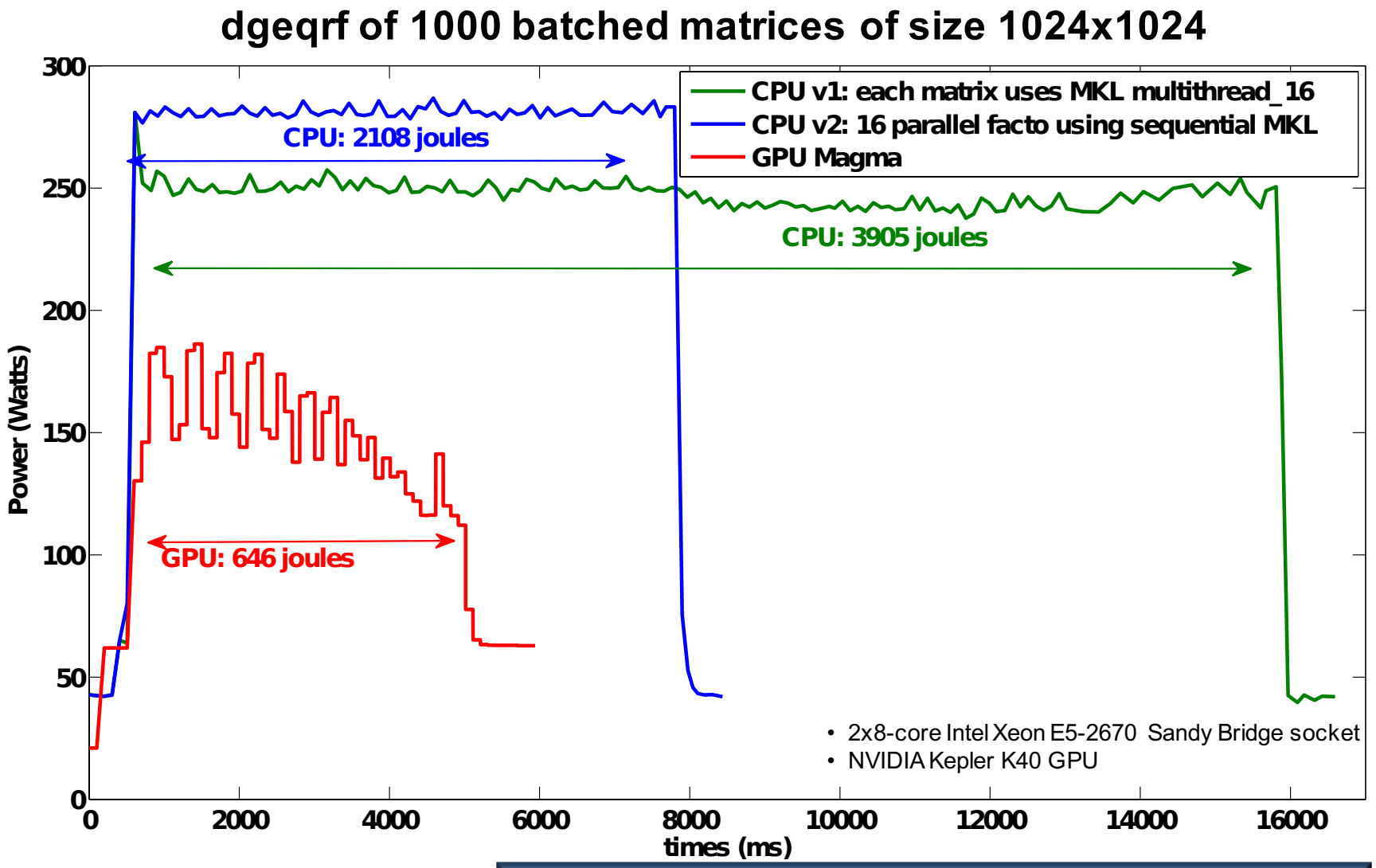
ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations Comparison to CPUs

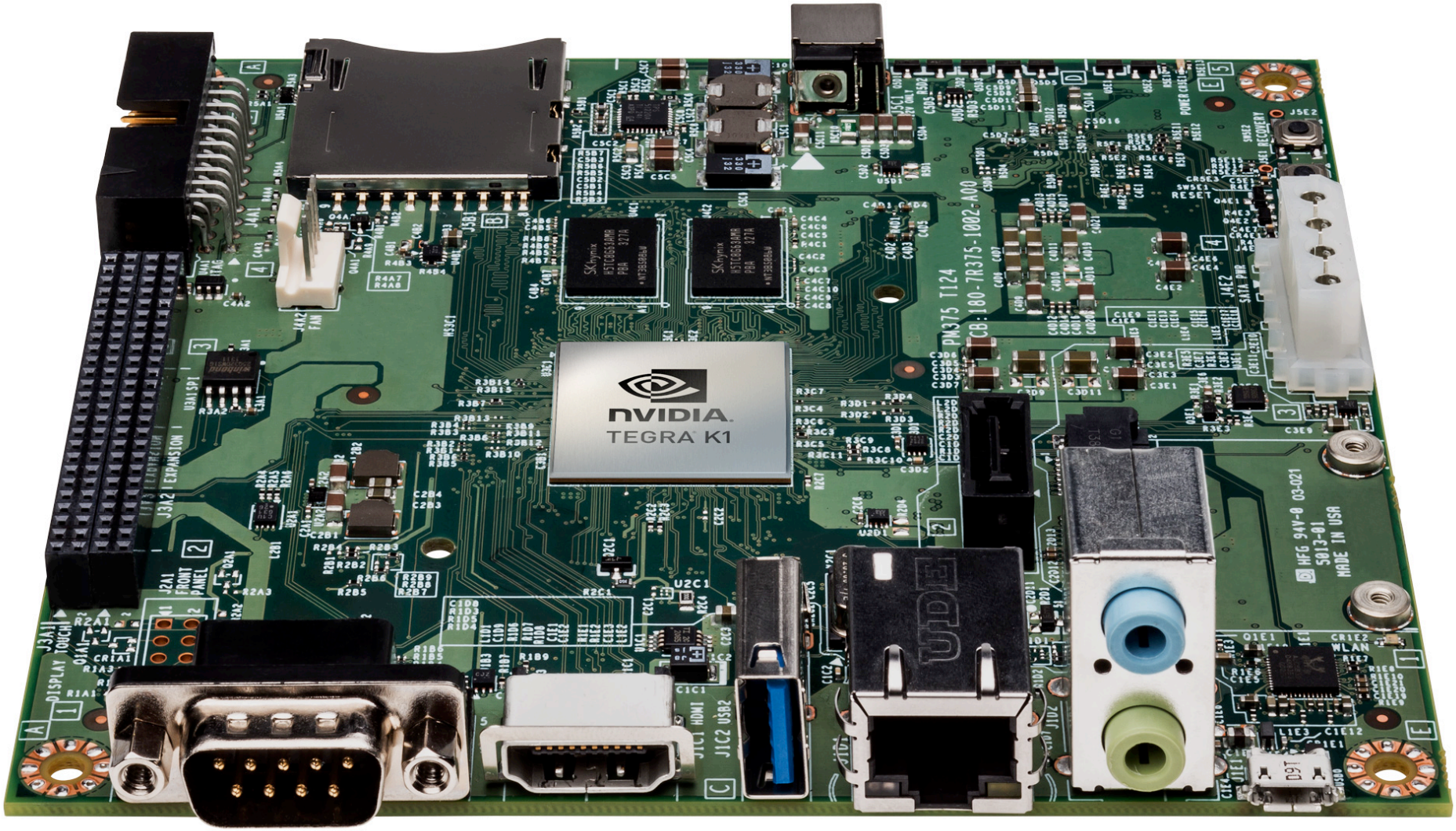


Energy efficiency GPU



CPU does not include DRAM power

NVIDIA Jetson TK1



ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

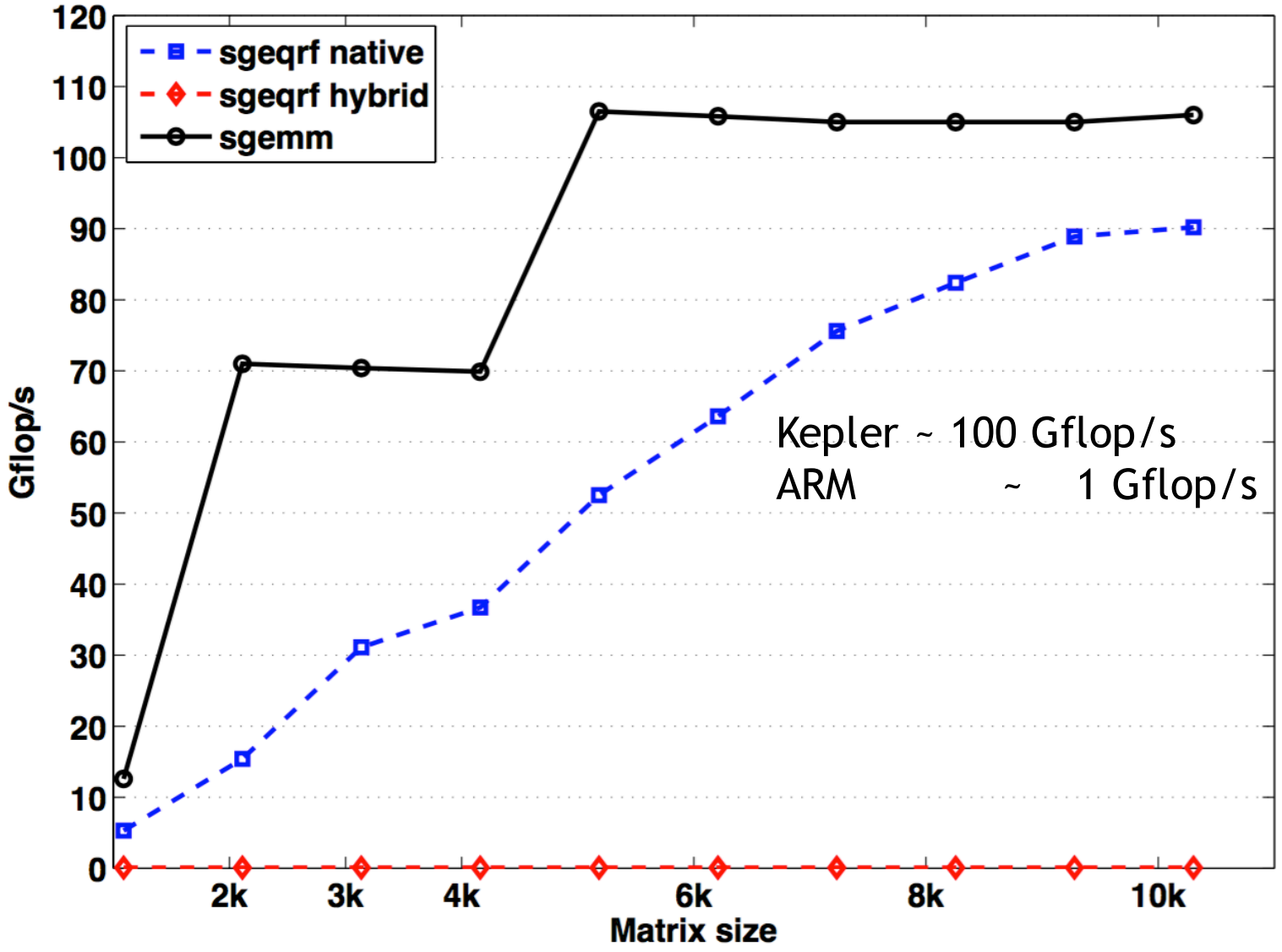
Tegra K1 Main Specs

GPU	
Architecture	Kepler
CUDA Cores	192
CPU	
Architecture	ARM Cortex A15 r3
Cores	4-plus-1
Frequency	2.3 GHz
Memory	
Type	DDR3L and LPDDR3
Size	8 GiB max (40 bit extension)
Display	
LCD	3840x2160
HDMI	4K (UltraHD, 4096x2160)
Manufacturing Process	28 nm

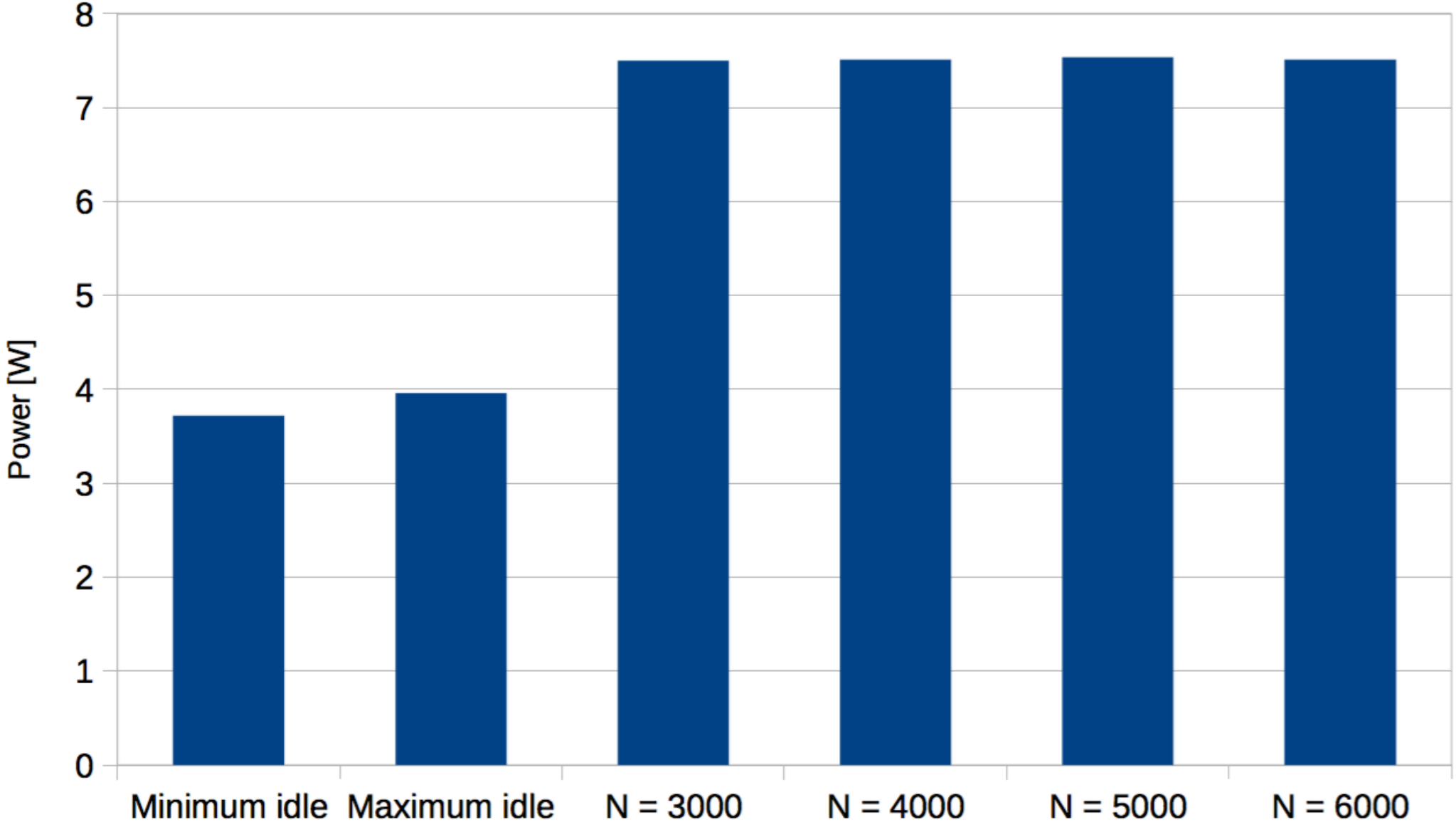
QR Factorization Highlights

- Perfect tool for imperfect data
- Works for over- and under-determined systems
- Deals with rank deficient matrices
  - Rank-revealing QR (RRQR)
  - Better stability than partial pivoting LU on some
- Parallelizes well
  - Numerical stability allows code optimization
- Reduces cost for SVD
  - For certain matrix shapes

Final Performance



Energy Summary on Jetson



ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations

Summary

- Batched computation can give a boost in performance for problem with very small sizes
- Traditional algorithmic design might not be the best direction
  - we need a new way of thinking
  - revisit and redesign algorithm to take advantage of the hardware specifics
- Performance modeling can help analyzing algorithm and their implementation, for example
  - An optimized GPU function cannot be efficient for all kind of computation, it depend on the context used for
  - Small computation are delicate and requires specific kernels (building block or fused).
  - Low level API is required to avoid overhead and context switching

Future Directions

- Extended functionality
  - Variable sizes (work in progress)
  - Mixed-precision techniques
  - Sparse direct multifrontal solvers & preconditioners
  - Applications
- Further tuning
  - autotuning
- GPU-only algorithms and implementations
- MAGMA Embedded

References

1. **A. Haidar, S. Tomov, A. Abdelfattah, M. Guidry, J. Billings, and J. Dongarra,**
   Optimisation Techniques Toward Accelerating Explicit Integration for Large Kinetic Networks.
   Submitted to the 45rd International Conference on Parallel Processing, Philadelphia, PA, USA ICPP 2016.

2. **A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, Tz. Kolev, I. Masliah, S. Tomov,**
   High-Performance Tensor Contractions for GPUs,
   The International Conference on Computational Science ICCS 2016.

3. **A. Abdelfattah, A. Haidar, S. Tomov,**
   Performance, Design, and Autotuning of Batched GEMM for GPUs,
   The International Supercomputing Conference IEEE-ISC 2016, Frankfurt, Germany.

4. **A. Haidar, S. Tomov, P. Luszczek, and J. Dongarra.**
   MAGMA Embedded: Towards a Dense Linear Algebra Library for Energy Efficient Extreme Computing
   IEEE High Performance Extreme Computing Conference IEEE-HPEC 2015, Waltham, MA USA.
   Best paper 2015

5. **A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra.**
   A Framework for Batched and GPU-resident Factorization Algorithms Applied to Block Householder Transformations
   International Supercomputing Conference IEEE-ISC 2015, Frankfurt, Germany.
   Springer, Lecture Notes in Computer Science Volume 9137, 2015, pp 31-47.

6. **K. Kabir, A. Haidar, S. Tomov, and J. Dongarra**
   On the Design, Development and Analysis of Optimized Matrix-Vector Multiplication Routines for coprocessors.
   International SuperComputing Conference IEEE-ISC 2015, Frankfurt, Germany
   Springer, Lecture Notes in Computer Science Volume 9137, 2015, pp 58-73

7. **A. Haidar, T. Dong, P. Luszczek, S. Tomov and J. Dongarra.**
   Batched Matrix Computations on Hardware Accelerators Based on GPUs.
   International Journal of High Performance Computing Applications 2014.

8. **A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra.**
   Optimization for Performance and Energy for Batched Matrix Computations on GPUs
   20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,
   Workshop on General Purpose Processing Using GPUs. GPGPU/PPoPP 2015.

9. **T. Dong, A. Haidar, S. Tomov, and J. Dongarra.**
   A Fast Batched Cholesky Factorization on a GPU
   ICPP 2014, The 43rd International Conference on Parallel Processing 2014.

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

ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science