Batched Matrix Computations on Hardware Accelerators Based on GPUs

Stan Tomov

w/ Azzam Haidar, Ahmad Ahmad, Piotr Luszczek, and Jack Dongarra

Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

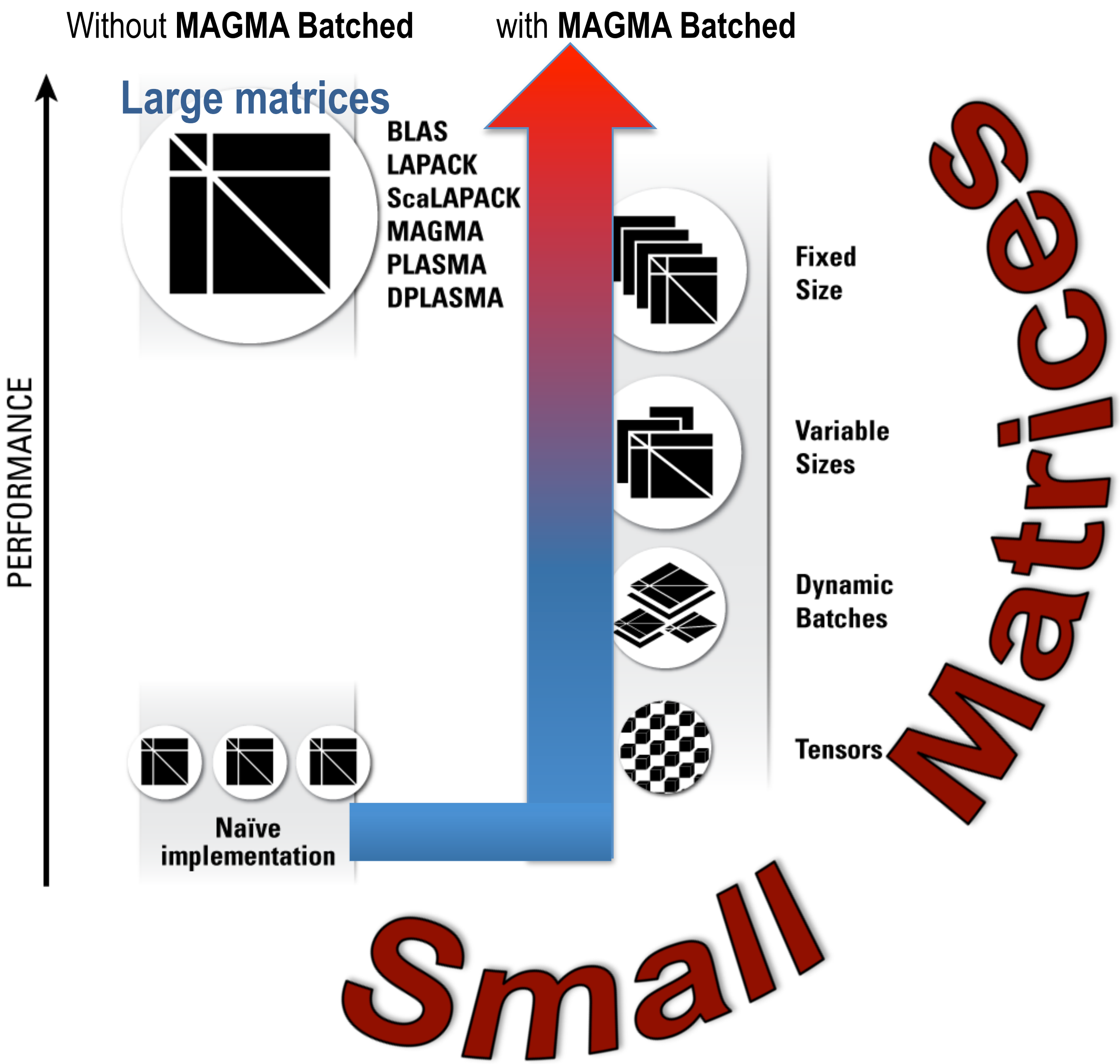
SIAM LA 2015
Atlanta, GA
October 29, 2015

Outline

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

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

Motivation



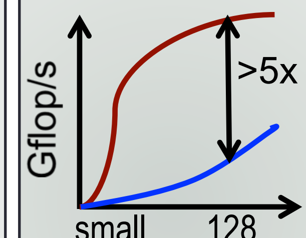
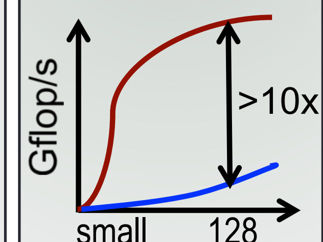
Linear Algebra on small problems are needed in many applications:

- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing,
  and more

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

Motivation ...

Batched *vs.* standard LA techniques

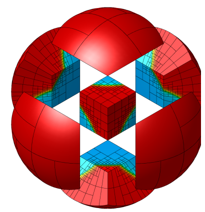
Techniques / LA problems	Batched (for small problems)	Standard (for large problems)	Expected acceleration ranges
Basic Linear Algebra Subprograms (BLAS)	**Batched BLAS** (no scheduling overheads)	Vendor optimized BLAS (e.g., CUBLAS, Intel MKL)	
Advanced routines: • Linear system solvers • Eigensolvers & SVD	• **Built on Batched BLAS** • **GPU-only** (no comm.) • **Batch-aware algorithms** • **Batch-scheduled**	• Built on BLAS • Hybrid CPU + GPU • High-level algorithms • DAG scheduling	

Examples

Need of **Tensor contractions** for **FEM simulations**

[collaboration with LLNL on BLAST package and Inria, France]

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

$$\textbf{Momentum Conservation:} \quad \frac{d\mathbf{v}}{dt} = -\mathbf{M_v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{de}{dt} = \mathbf{M_e}^{-1}\mathbf{F^T}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M_v}$ and $\mathbf{M_e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step.

[1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641. (36 pages)

- Contractions can often be implemented as index reordering plus **batched GEMM** (and hence, be highly efficient)

TENSOR KERNELS FOR ASSEMBLY/EVALUATION 3

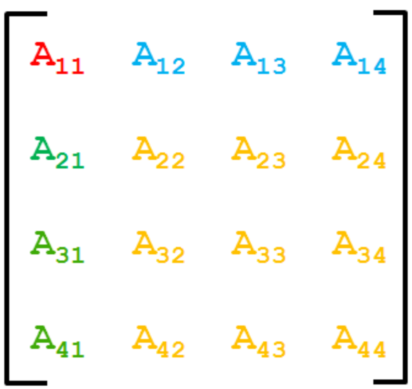
stored components	FLOPs for assembly	amount of storage	FLOPs for matvec	numerical kernels	
full assembly					
M	$O(p^{3d})$	$O(p^{2d})$	$O(p^{2d})$	$B, D \mapsto B^T DB,\ x \mapsto Mx$	
decomposed evaluation					
B, D	$O(p^{2d})$	$O(p^{2d})$	$O(p^{2d})$	$x \mapsto Bx,\ x \mapsto B^T x,\ x \mapsto Dx$	
near-optimal assembly – equations (1) and (2)					
$M_{i_1,\cdots,j_d}$	$O(p^{2d+1})$	$O(p^{2d})$	$O(p^{2d})$	$A_{i_1,k_2,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2}$	(1a)
				$A_{i_1,i_2,,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,j_1}$	(1b)
				$A_{i_1,k_2,k_3,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2,k_3}$	(2a)
				$A_{i_1,i_2,k_3,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,k_3,j_1}$	(2b)
				$A_{i_1,i_2,i_3,j_1,j_2,j_3} = \sum_{k_3} B^{1d}_{k_3,i_3} B^{1d}_{k_3,j_3} C_{i_1,i_2,k_3,j_1,j_2}$	(2c)
near-optimal evaluation (partial assembly) – equations (3) and (4)					
B^{1d}, D	$O(p^d)$	$O(p^d)$	$O(p^{d+1})$	$A_{j_1,k_2} = \sum_{j_2} B^{1d}_{k_2,j_2} V_{j_1,j_2}$	(3a)
				$A_{k_1,k_2} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2}$	(3b)
				$A_{k_1,i_2} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2}$	(3c)
				$A_{i_1,i_2} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2}$	(3d)
				$A_{j_1,j_2,k_3} = \sum_{j_3} B^{1d}_{k_3,j_3} V_{j_1,j_2,j_3}$	(4a)
				$A_{j_1,k_2,k_3} = \sum_{j_2} B^{1d}_{k_2,j_2} C_{j_1,j_2,k_3}$	(4b)
				$A_{k_1,k_2,k_3} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2,k_3}$	(4c)
				$A_{k_1,k_2,i_3} = \sum_{k_3} B^{1d}_{k_3,i_3} C_{k_1,k_2,k_3}$	(4d)
				$A_{k_1,i_2,i_3} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2,i_3}$	(4e)
				$A_{i_1,i_2,i_3} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2,i_3}$	(4f)
matrix-free evaluation					
none	none	none	$O(p^{d+1})$	evaluating entries of B^{1d}, D, (3a)–(4f) sums	

Examples

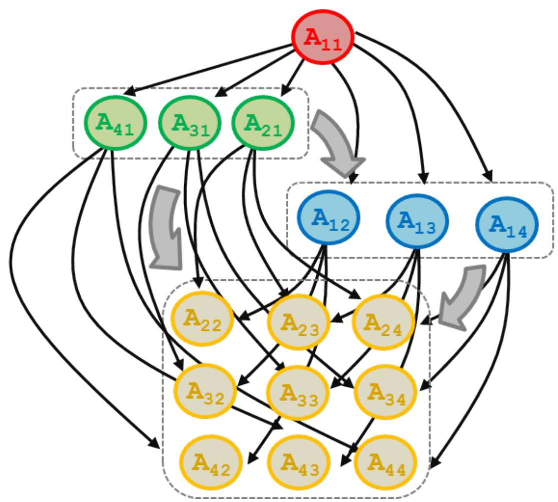
Need of **Batched** routines for **Numerical LA**

[e.g., sparse direct multifrontal methods, preconditioners for sparse iterative methods, tiled algorithms in dense linear algebra, etc.;]

Sparse / Dense Matrix System

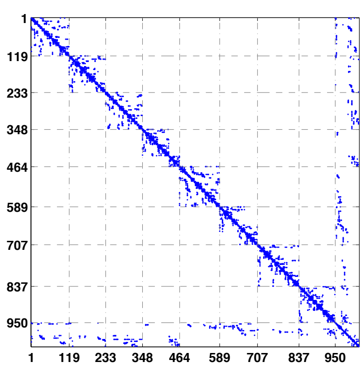


DAG-based factorization



To capture main LA patterns needed in a numerical library for Batched LA

- **LU, QR,** or **Cholesky** on small diagonal matrices
- **TRSMs**, **QRs**, or **LUs**
- **TRSMs**, **TRMMs**
- **Updates (Schur complement) GEMMs**, **SYRKs**, **TRMMs**



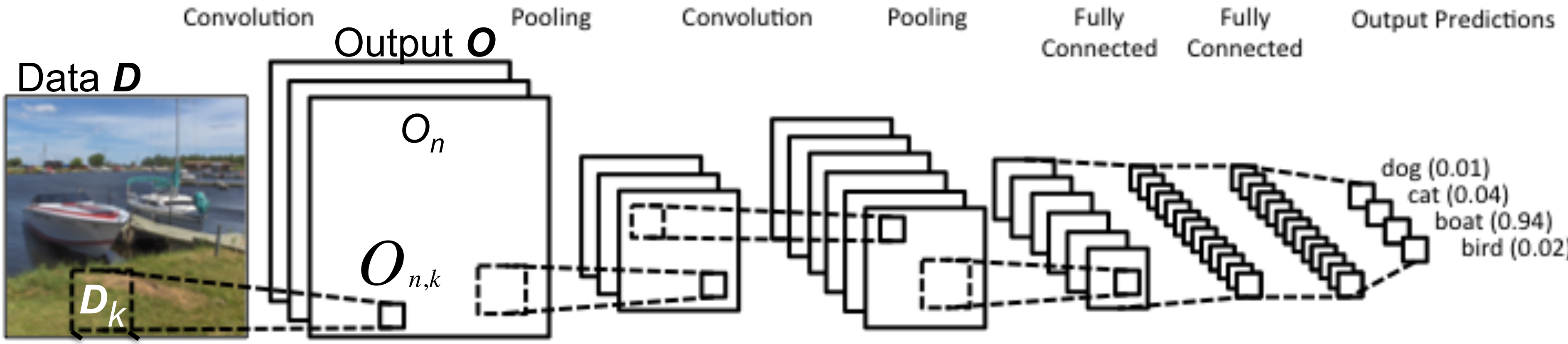
- Example matrix from Quantum chromodynamics
- Reordered and ready for sparse direct multifrontal solver
- Diagonal blocks can be handled in parallel through batched LU, QR, or Cholesky factorizations

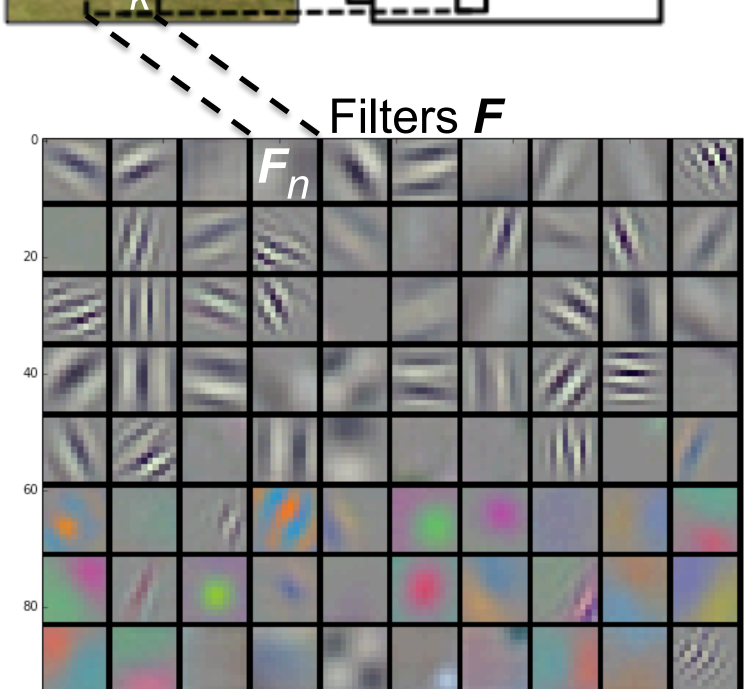
ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Examples

Need of **Batched and/or Tensor contraction** routines in **machine learning**

e.g., Convolutional Neural Networks (CNNs) used in computer vision
Key computation is convolution of Filter Fi (feature detector) and input image D (data):





Convolution operation:

- For every filter F_n and every channel, the computation for every pixel value $O_{n,k}$ is a **tensor contraction**:

$$O_{n,k} = \sum_i D_{k,i} F_{n,i}$$

- Plenty of parallelism; small operations that must be batched
- With data "reshape" the computation can be transformed into a **batched GEMM** (and hence, efficiently implemented; among other approaches)

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

Examples

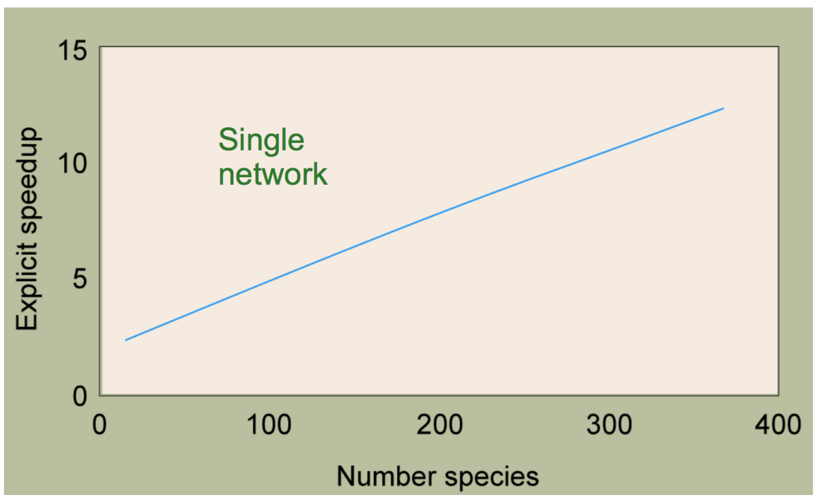
Multi-physics problems need Batched LA on small problems

Collaboration with ORNL and UTK physics department (Mike Guidry, Jay Billings, Ben Brock, Daniel Shyles, Andrew Belt)

- Many physical systems can be modeled by a fluid dynamics plus kinetic approximation e.g., in astrophysics, stiff equations must be integrated numerically:
  - **Implicitly**; standard approach, leading to need of batched solvers (e.g., as in XNet library)
  - **Explicitly**; a new way to stabilize them with Macro- plus Microscopic equilibration

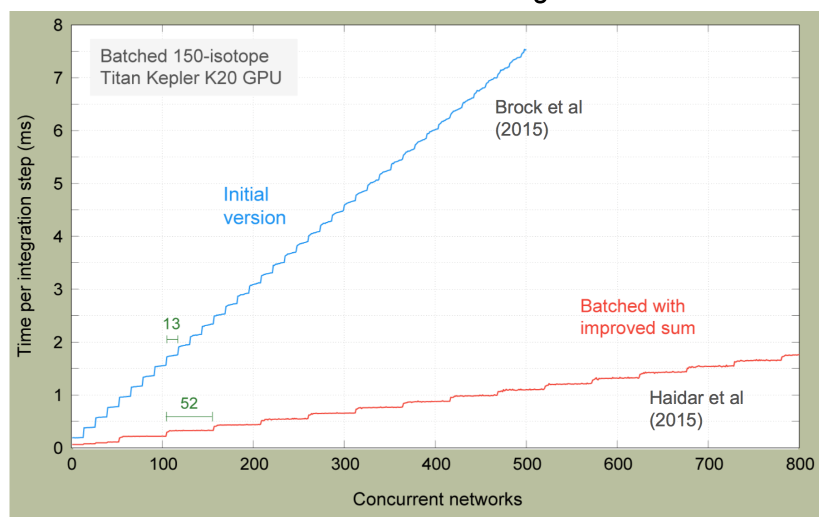
need batched tensor contractions of variable sizes

Explicit vs. Implicit speedup on single network



10x speedup on few hundred species
(few hundred dof batched solve in implicit methods)

Additional acceleration achieved through MAGMA Batched



An additional **7x speedup** over initially highly optimized explicit method implementation

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

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 HP linear algebra software for small problems** especially for GPU
- **CPU**: this can be done easily using existing software infrastructure
- **GPU**: are efficient for large data parallel computations, and therefore have often been used in combination with CPUs, where the CPU handles the small and difficult tasks to be parallelized
- What programming model is best for small problems?

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

MAGMA Batched Computations

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

Our goal:

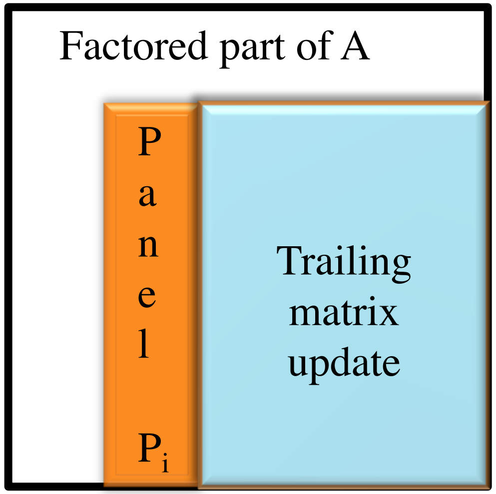
- Develop a high-performance numerical library for batched linear algebra subroutines tuned for performance and energy efficiency on modern processor architectures
- Consider hardware specifics – the higher ratio of execution and the memory model – of the new & emerging accelerators and coprocessors
- 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]

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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

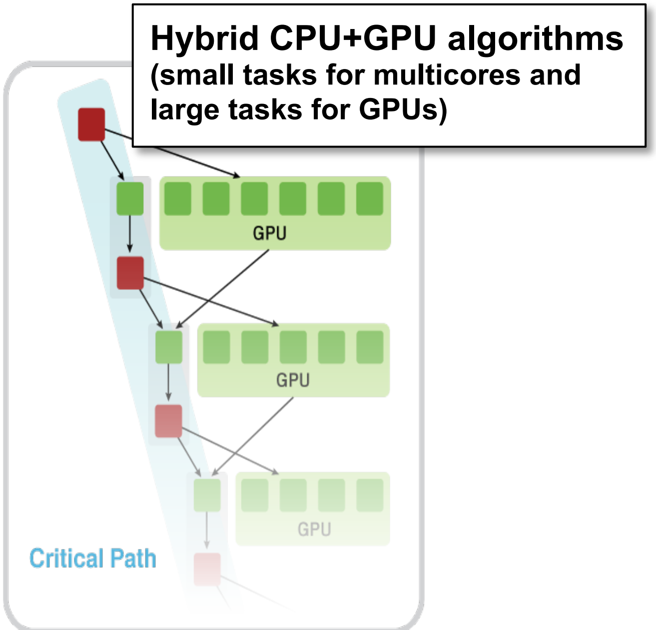


Hardware characteristics and limitations to consider:

- GPU memory is limited (48KB of shared per SMX, limited number of register)
- Prefer implementation that extensively uses large number of thread/block (a warp is 32 threads)
- Prefer coalescent memory access (32 threads can read in parallel 32 elements)

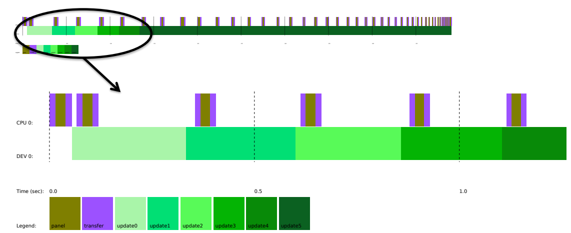
ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Approach



Classical strategies design

- For large problems the strategy is to prioritize the data-intensive operations to be executed by the accelerator and keep the small (often 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**

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

MAGMA Batched Approach

Classical strategies design

- For large stand-alone problems performance is driven by the update operations

Challenges

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

Proposition

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

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

MAGMA Batched low-level strategies

Classical strategies design

- A recommended way of writing efficient GPU kernels is to use the GPU's shared memory – 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 low-level strategies

Challenges

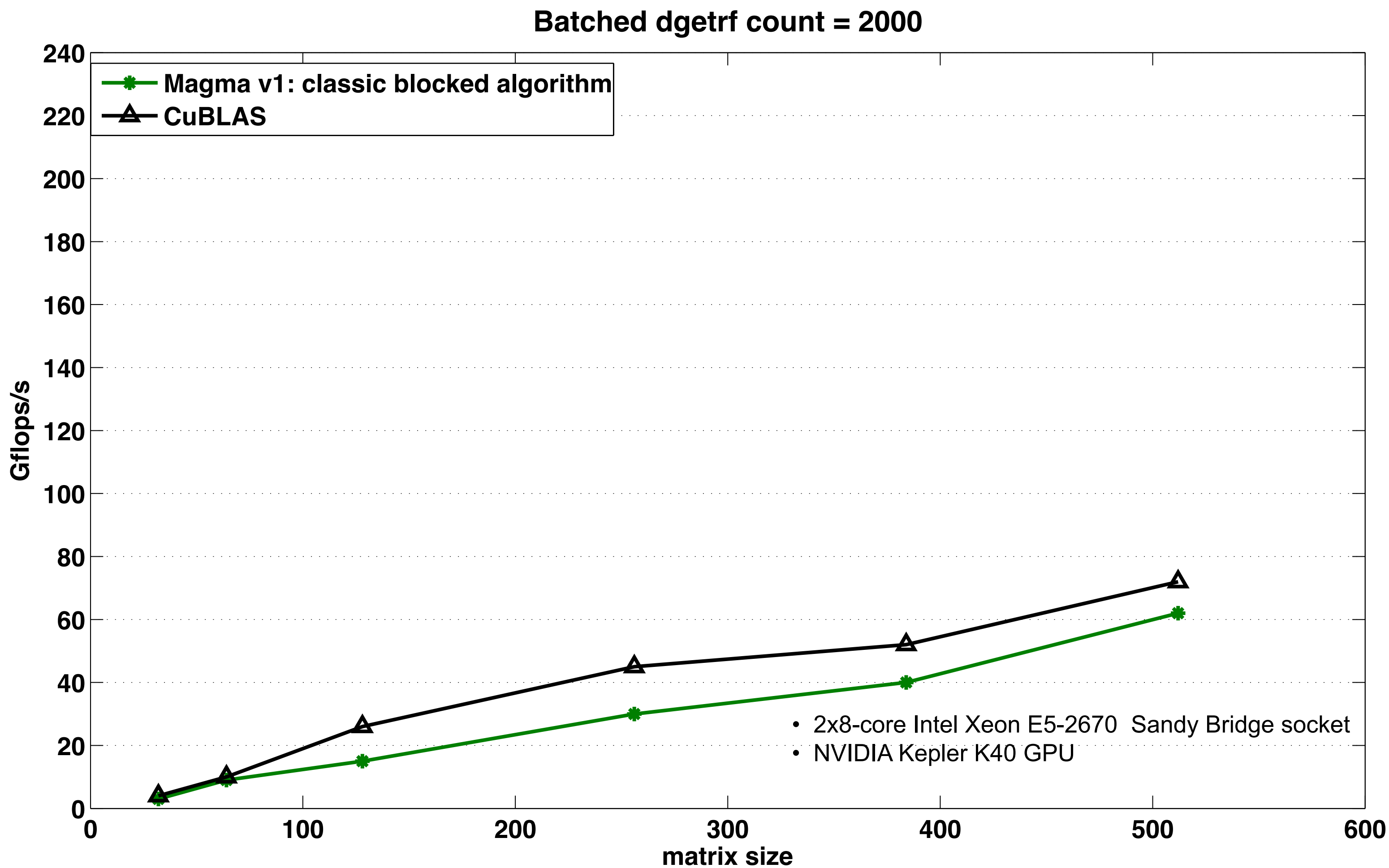
- Completely saturating the shared memory per SMX can decrease the performance of memory bound operations, since only one thread-block will be mapped to that SMX at a time (low occupancy)
- due to a limited parallelism in the panel computation, the number of threads used in the thread block will be limited,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

Proposition: custom design per operations type

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

MAGMA Batched Computations

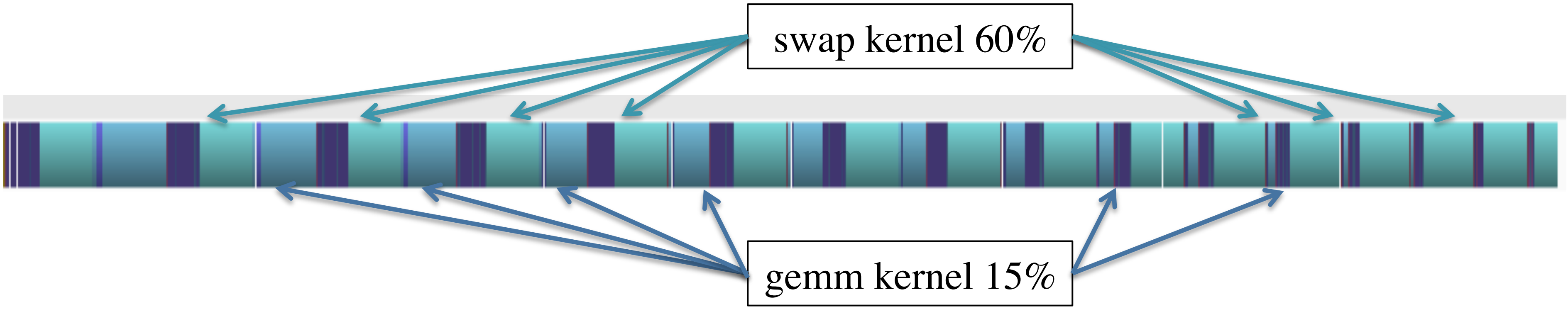
Consider the LU factorization



ICL INNOVATIVE COMPUTING LABORATORY

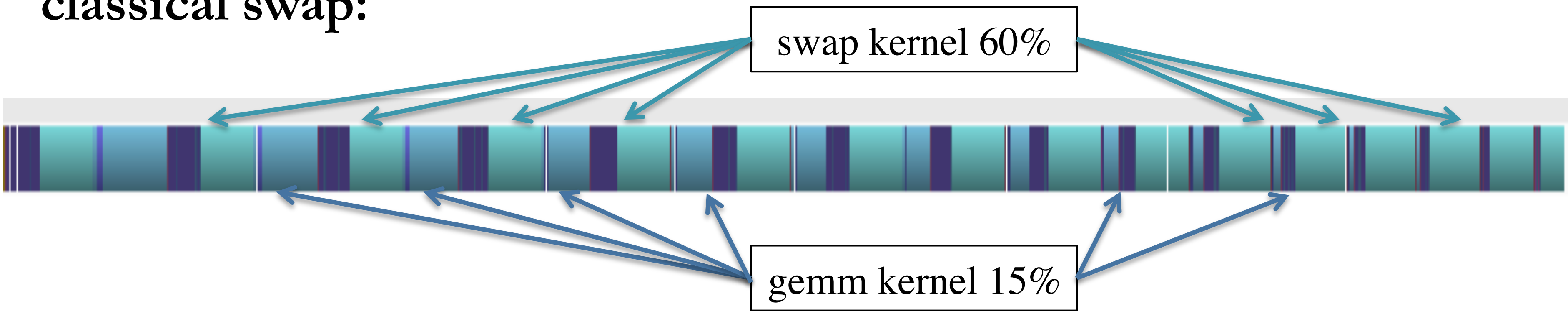
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Profile and trace to find bottlenecks

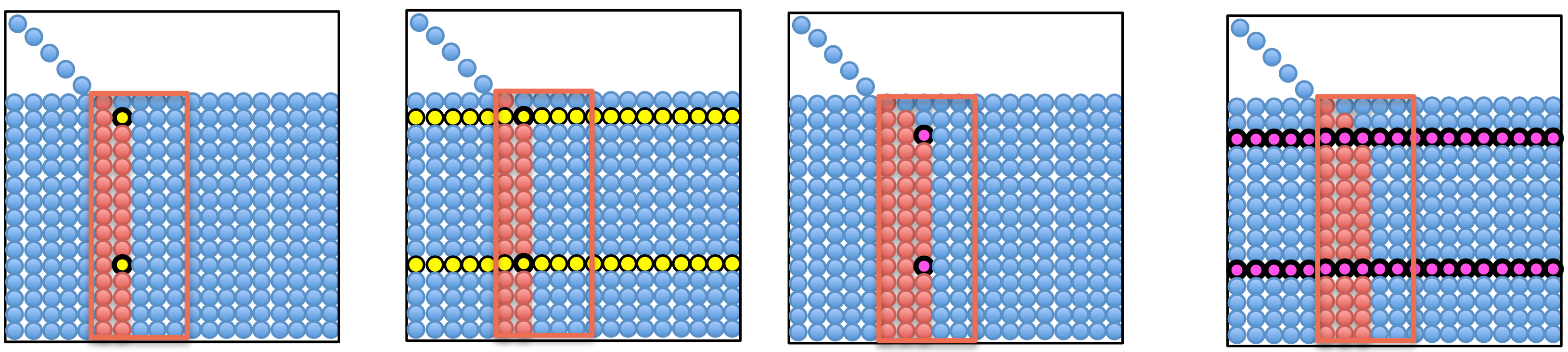


ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

classical swap:

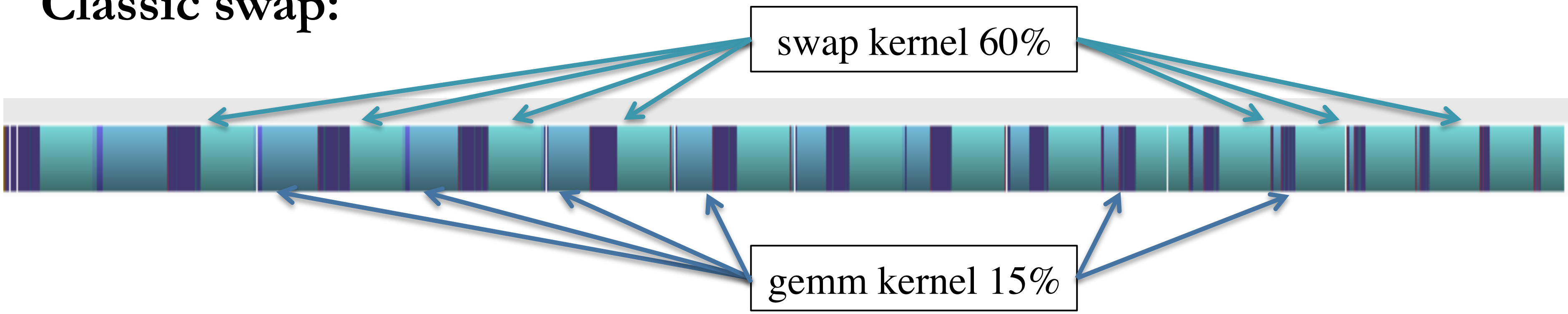


How does the swap work?

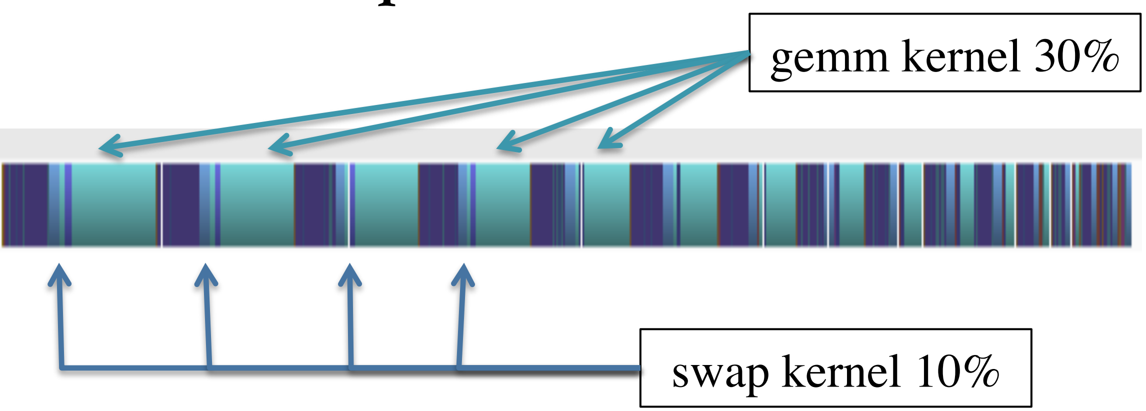


ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Classic swap:

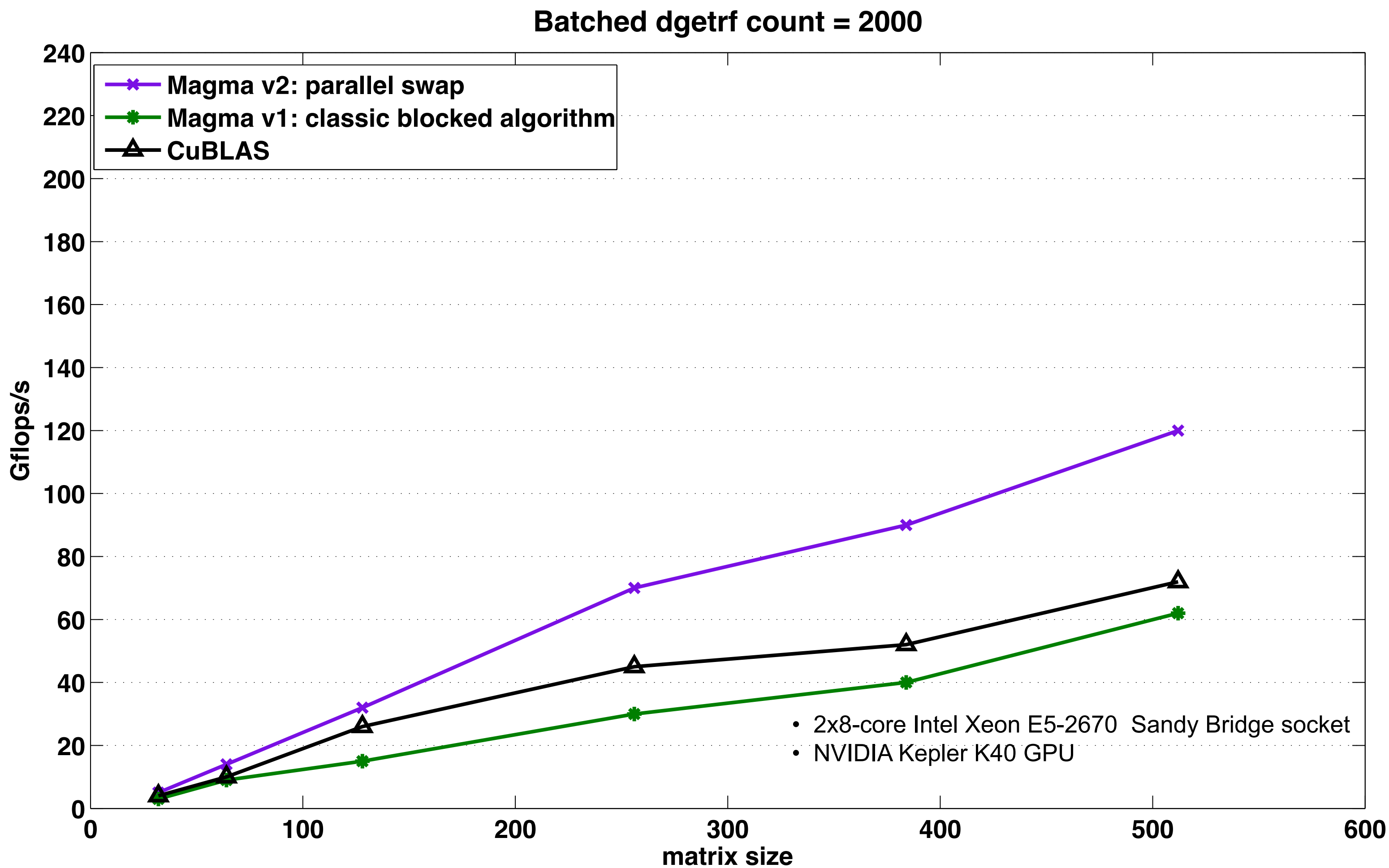


Parallel swap:



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

MAGMA Batched Computations

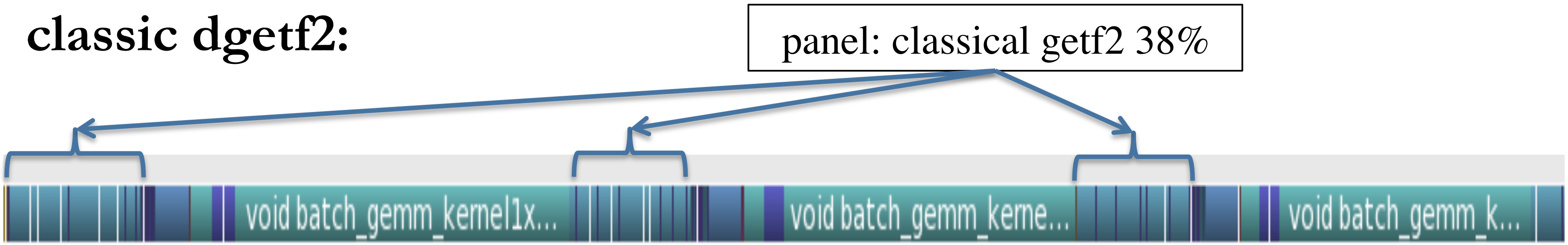


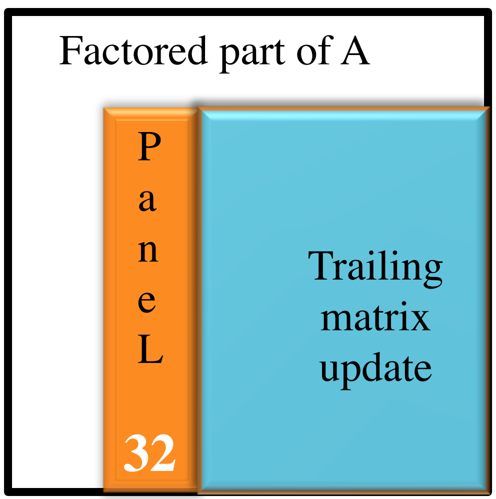
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations

Panel factorization classic dgetf2:





Bottlenecks:

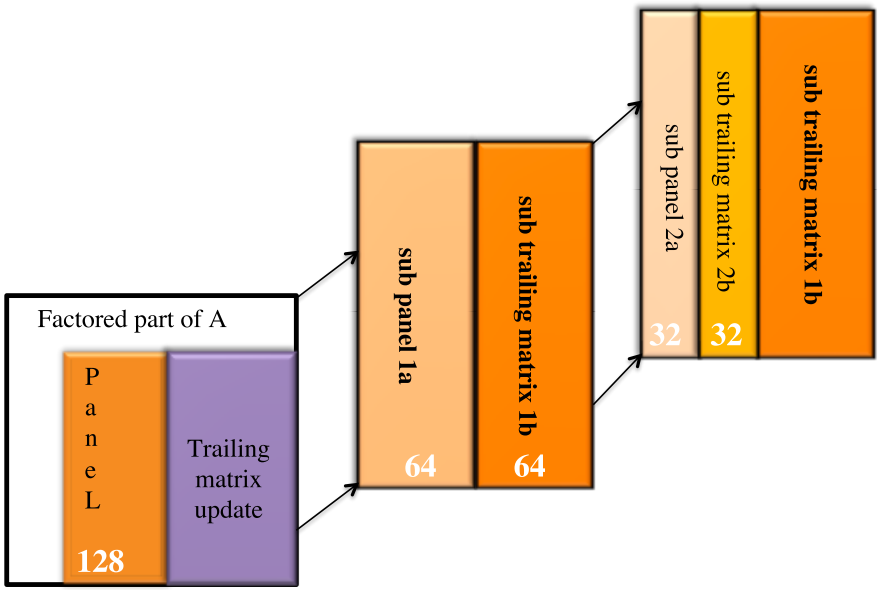
- *nb* large: panel get slower
  --> very bad performance.
- *nb* small: panel get faster but the update is not anymore efficient since dealing with gemm's of small sizes
  --> very bad performance.
- trade-off ? No effect, since we are talking about small size.

Proposition:

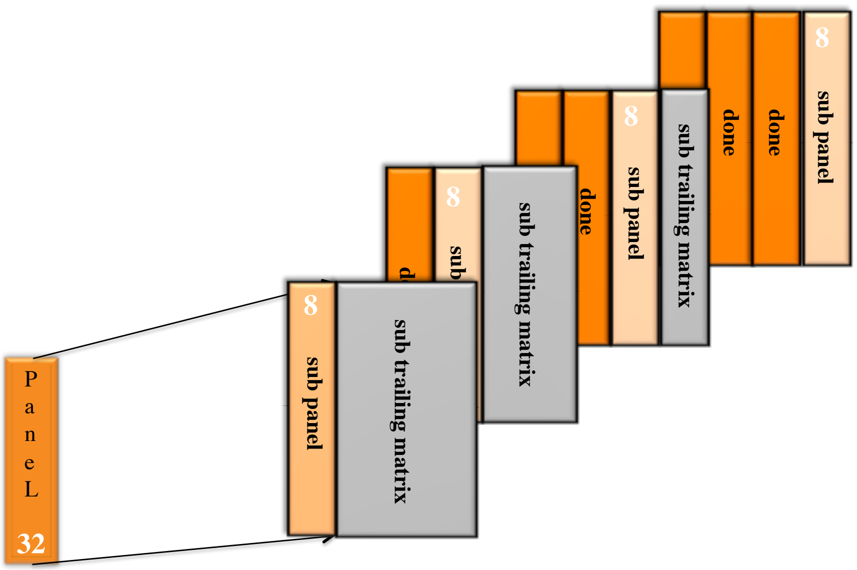
- We propose to develop **two layers blocking**: a recursive and nested blocking technique that block also the panel.

MAGMA Batched Computations

Two-layers blocking:



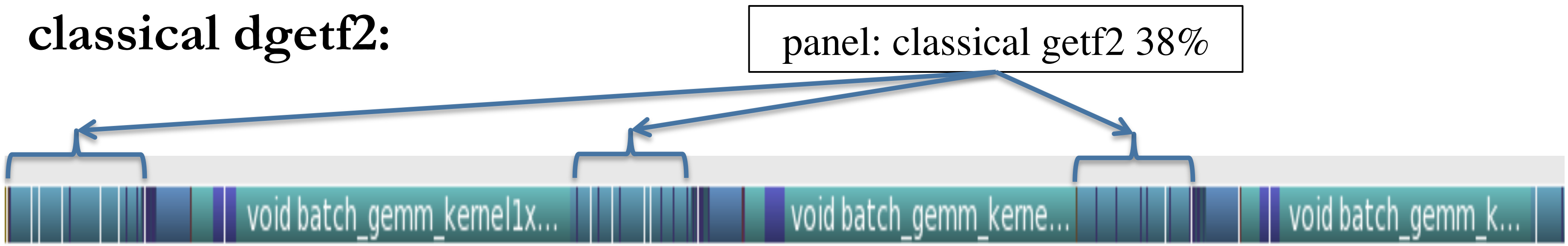
(a) Recursive nested blocking fashion.



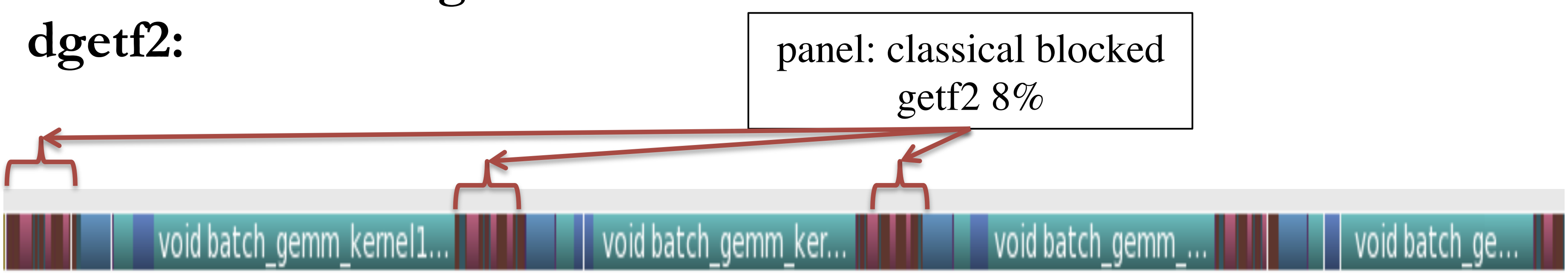
(b) Classical blocking fashion.

MAGMA Batched Computations

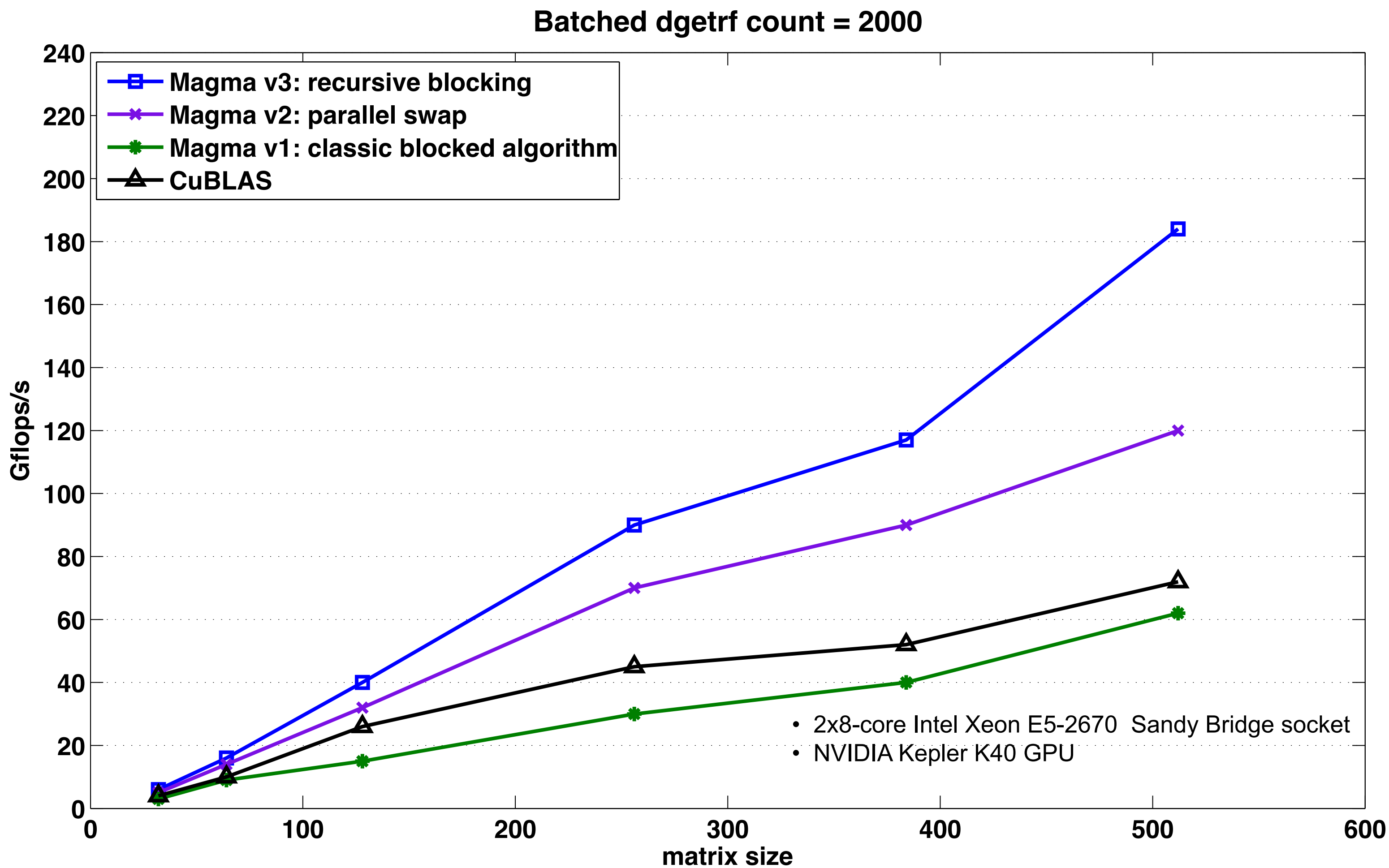
panel factorization classical dgetf2:



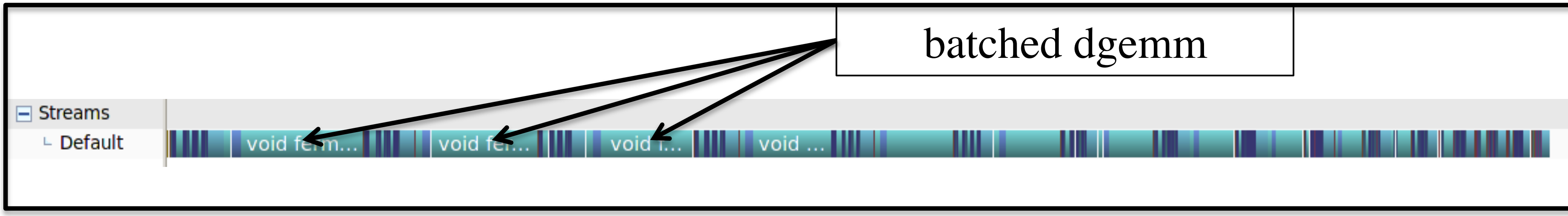
Recursive blocking of dgetf2:



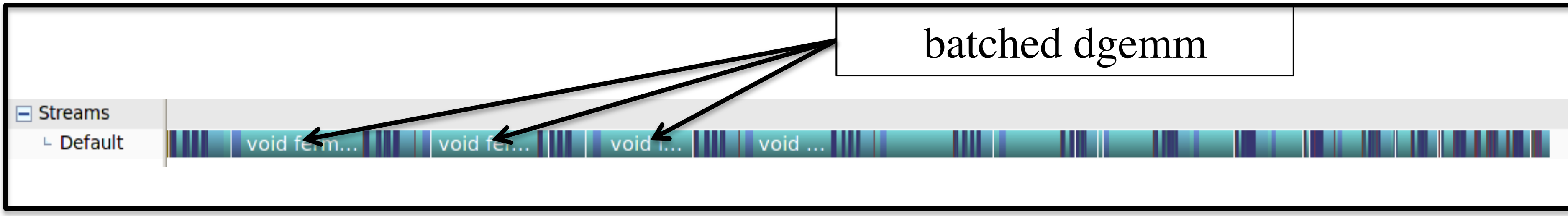
MAGMA Batched Computations

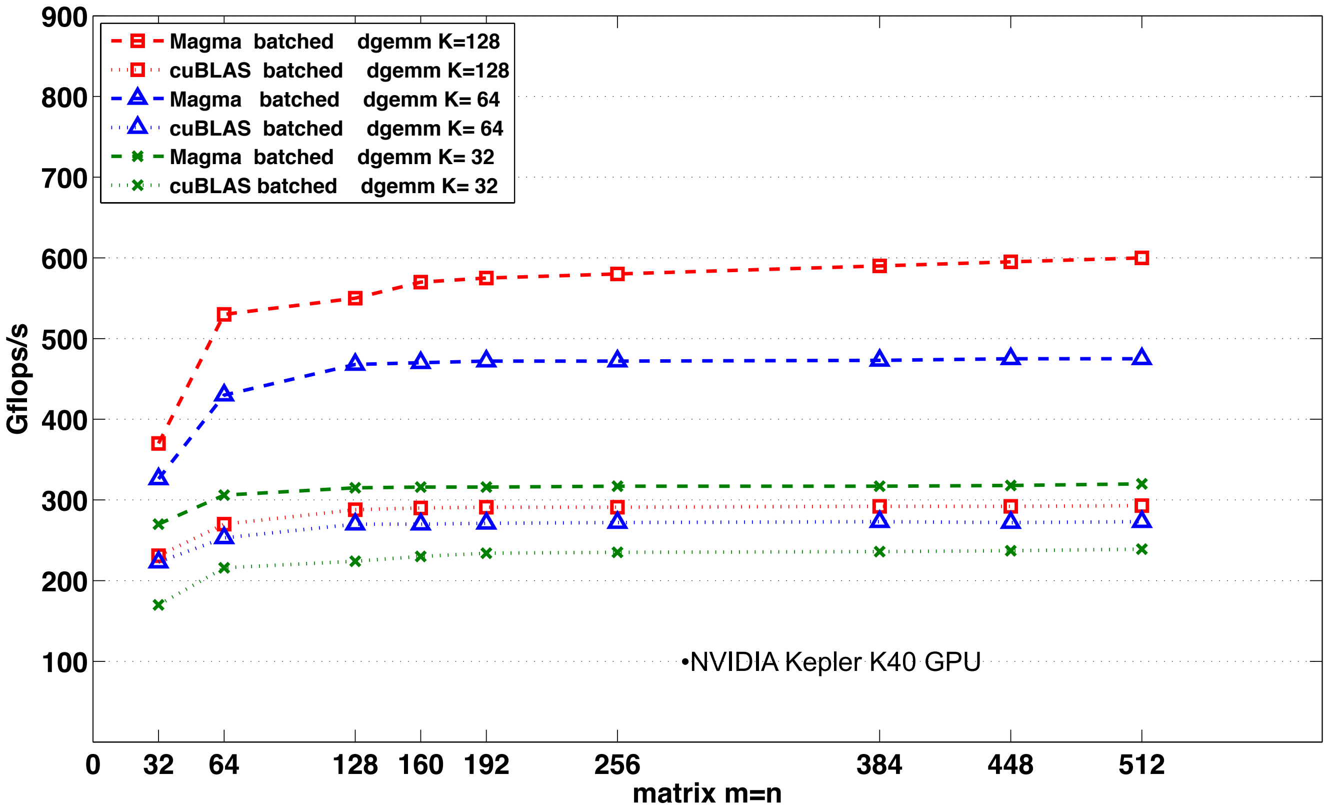


MAGMA Batched Computations



MAGMA Batched Computations

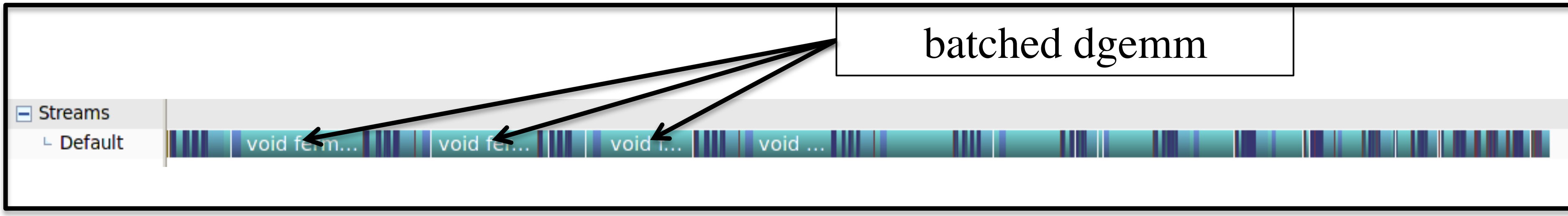


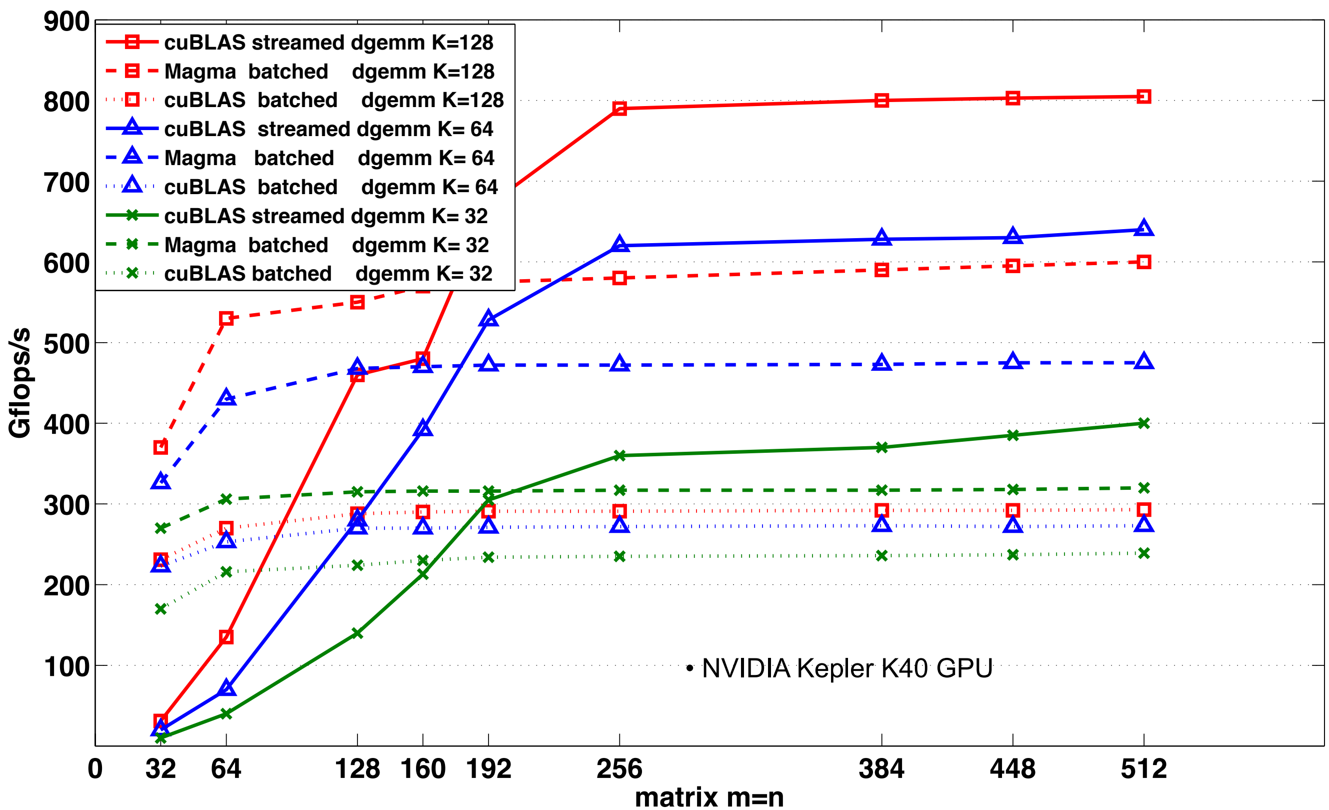


ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

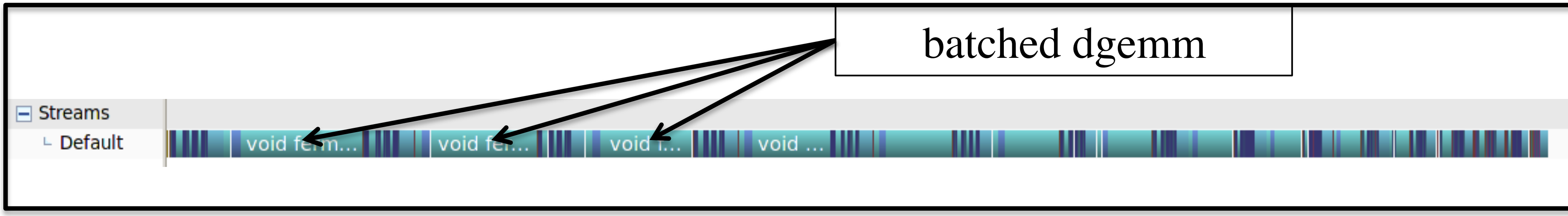
MAGMA Batched Computations

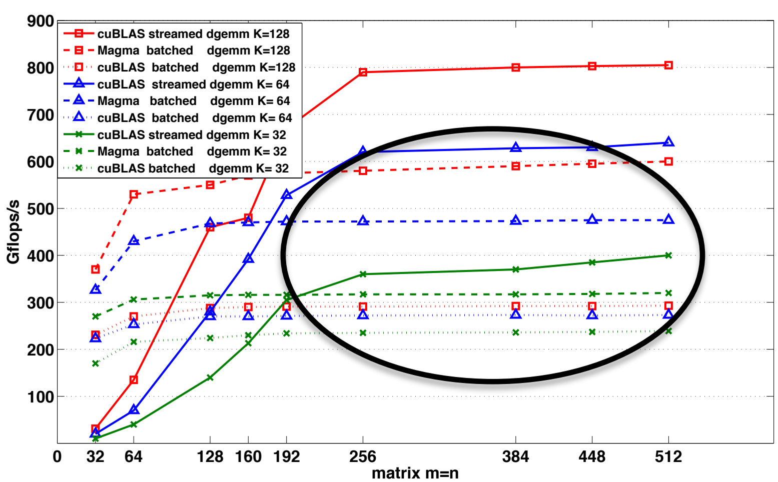




ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations





Bottlenecks:

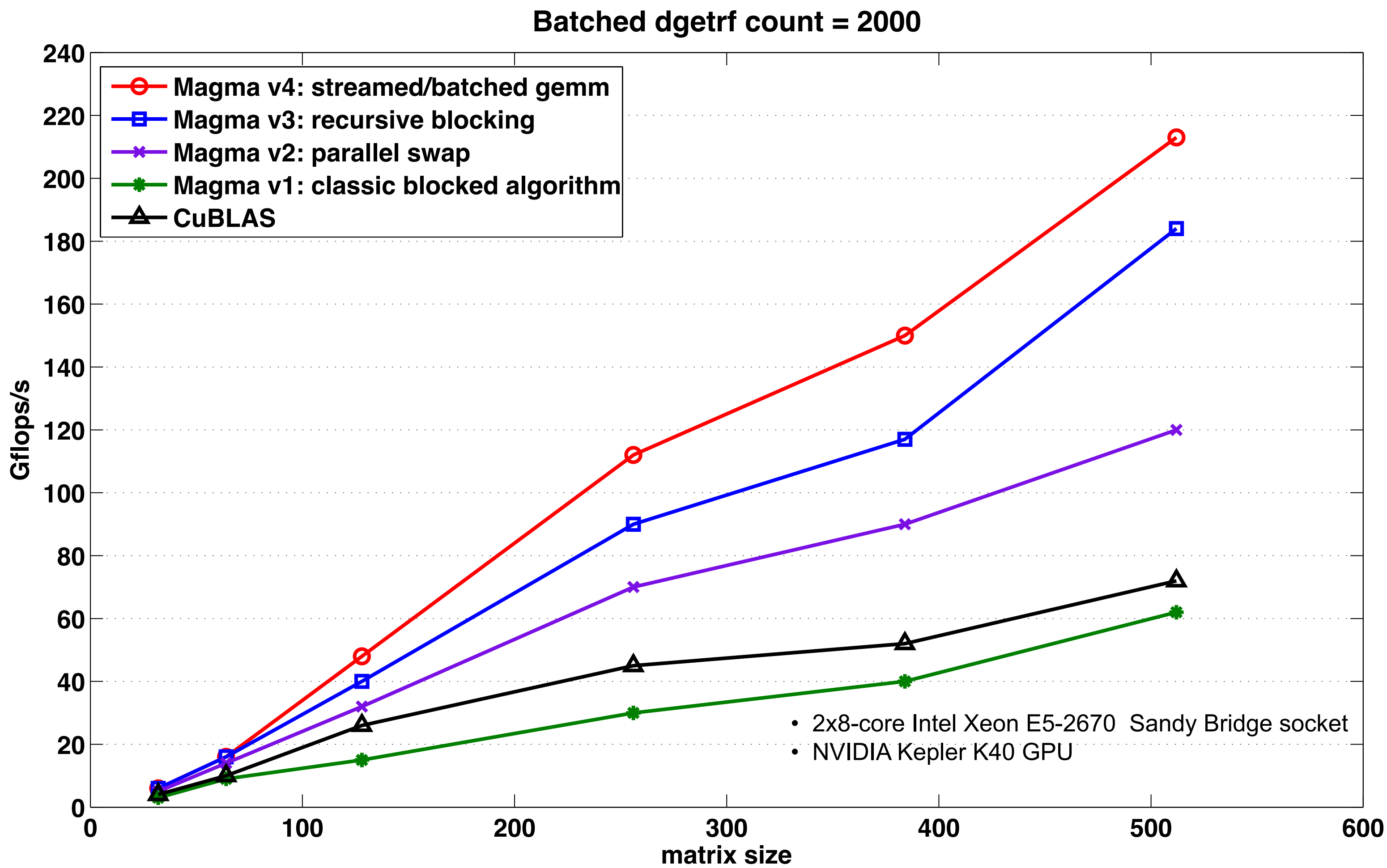
- Batched gemm kernel from cuBLAS and Magma are well suited for small matrix sizes (128) but stagnate for larger sizes (>128)

Proposition:

- Streamed gemm can provide higher performance for large matrix size (>128) and thus we propose to use both streamed and batched according to the size of the trailing matrix

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

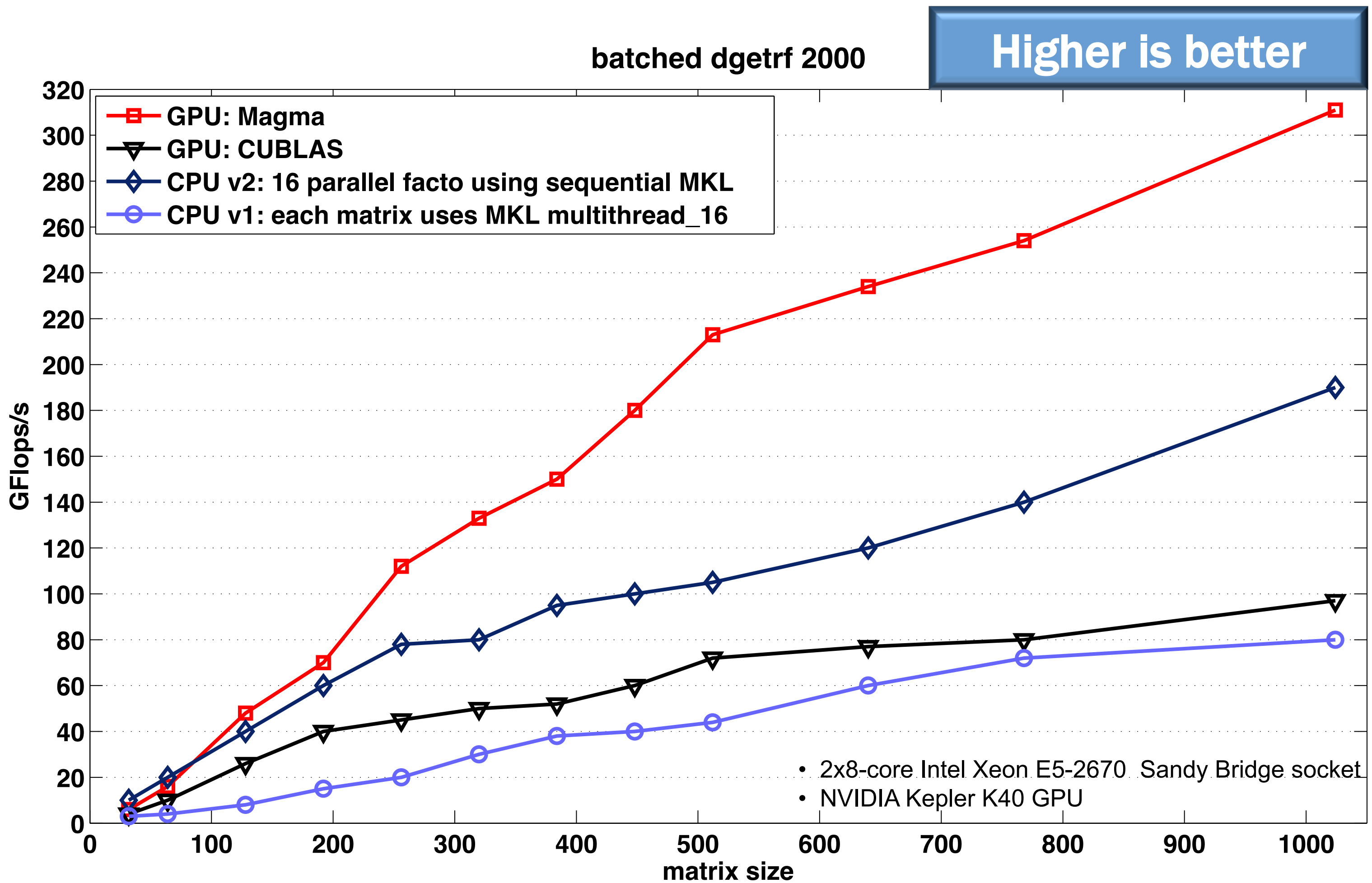
MAGMA Batched Computations



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

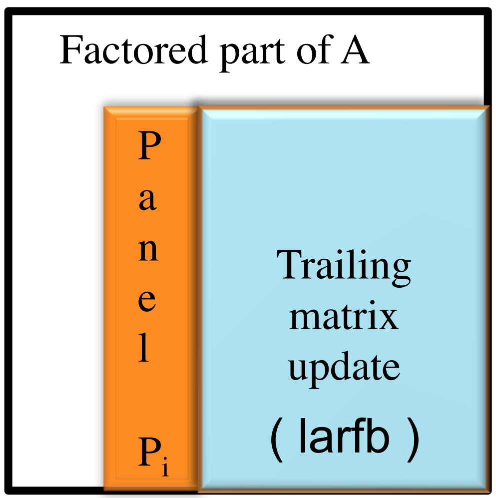
MAGMA Batched Computations Comparison to CPUs



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

MAGMA Batched QR

- **Similar design and optimization methodology**

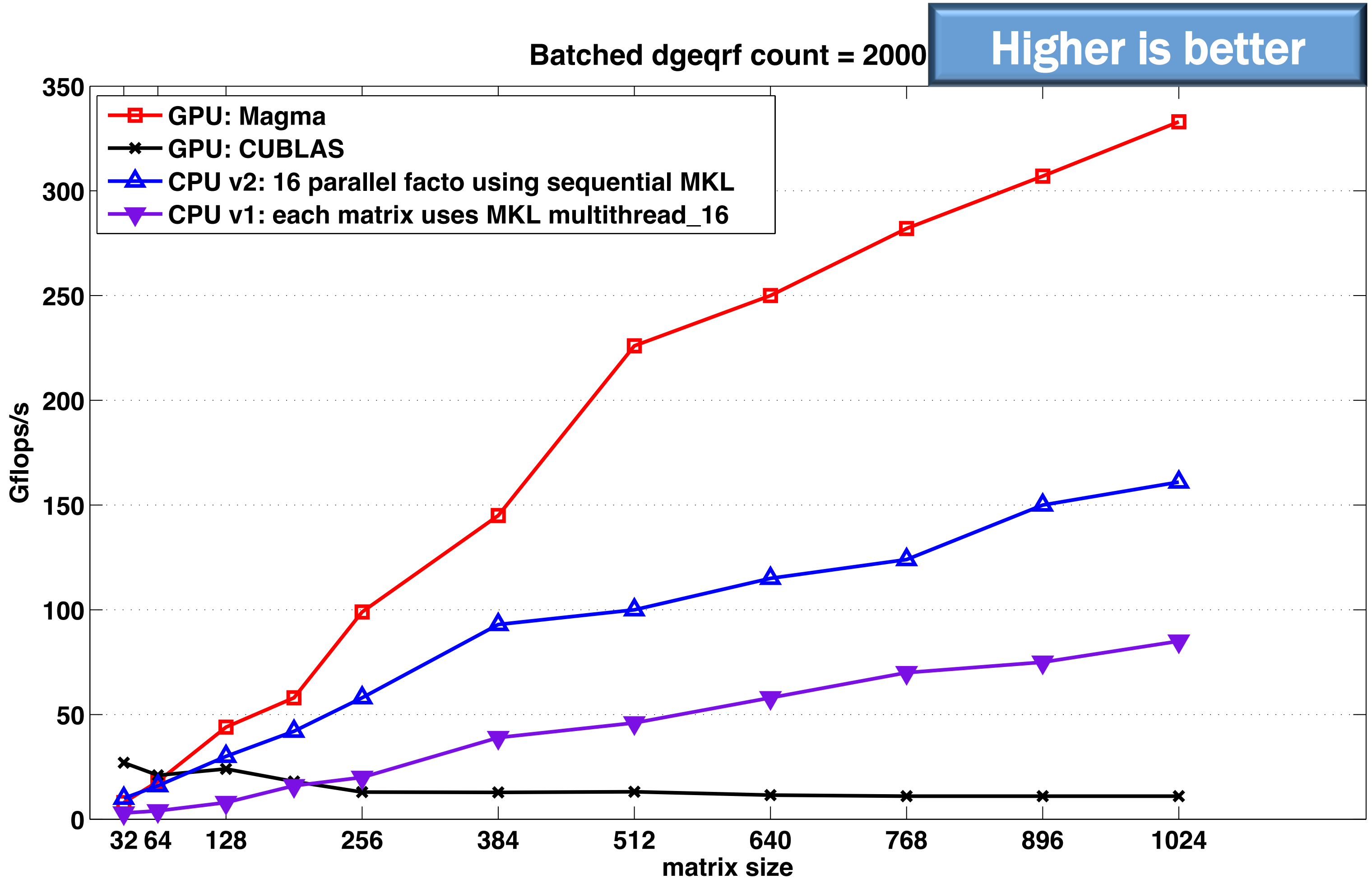


- Panel is recursive
- GEMMs in the update are similarly optimized and tuned
- Matrix update – apply $(I - V_i T_i V_i^T)$ to the trailing matrix
  - T is triangular; computed column-by column (larft); memory bound; takes 50% of total factorization time

    for $j \in \{1, 2, \ldots, nb\}$ **do**
    - dgemv to compute $\widehat{T}_{1:j-1,j} = A^H_{j:m,1:j-1} \times A_{j:m,j}$;
    - dtrmv to compute $T_{1:j-1,j} = T_{1:j-1,1:j-1} \times \widehat{T}_{1:j-1,j}$;
    - $T(j,j) = tau(j)$;
  - Computation of T is replaced by a new Blocked algorithm leading to 20-30% speedup
- Extra flops for higher performance (not all flops are =)
  - T (upper triangular) is filled up with 0s in lower part and used with gemm (instead of trmm), bringing ~10% speedup

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

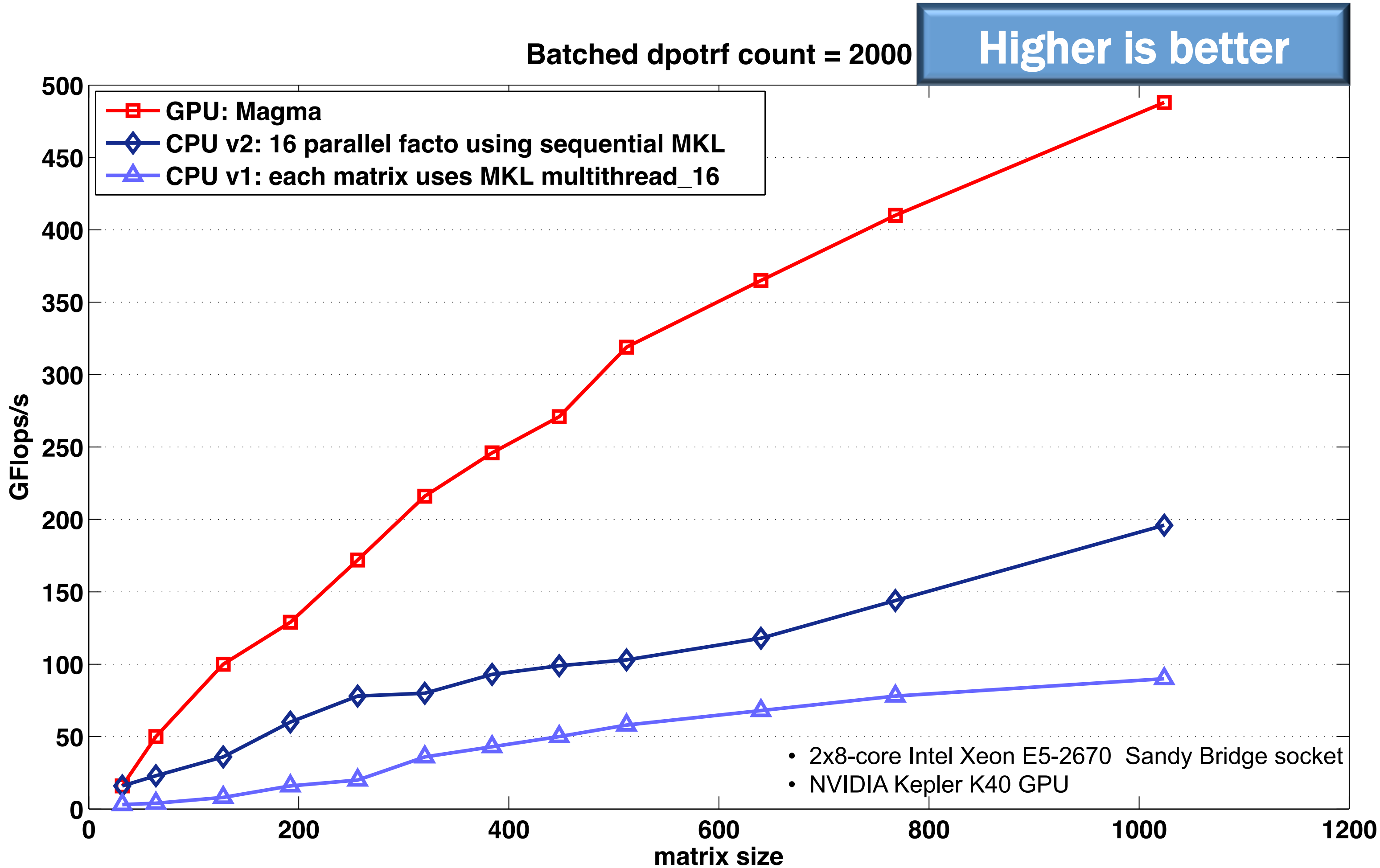
MAGMA Batched QR



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

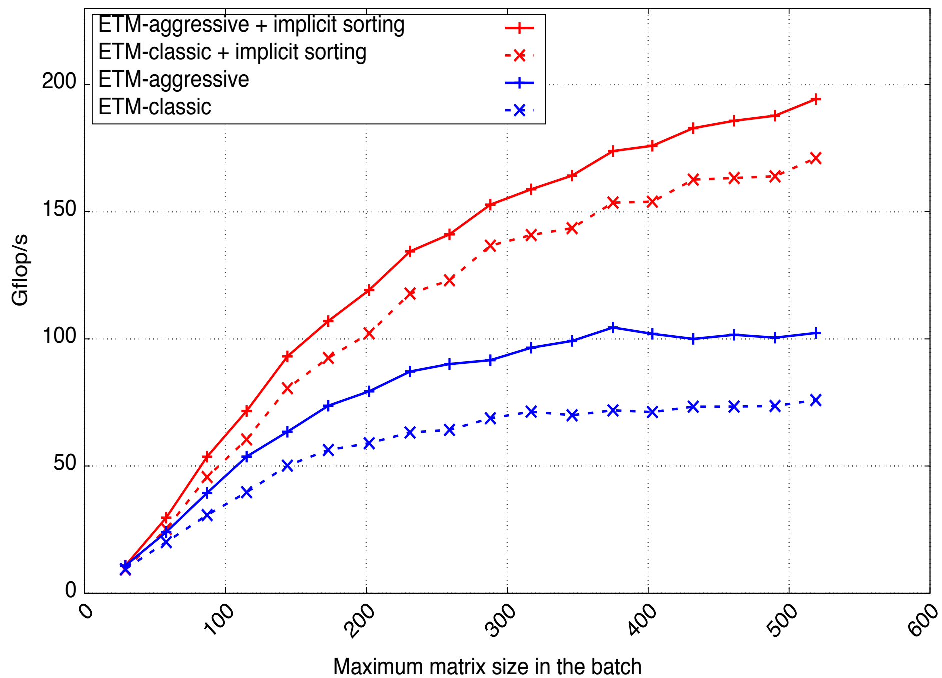
ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Cholesky

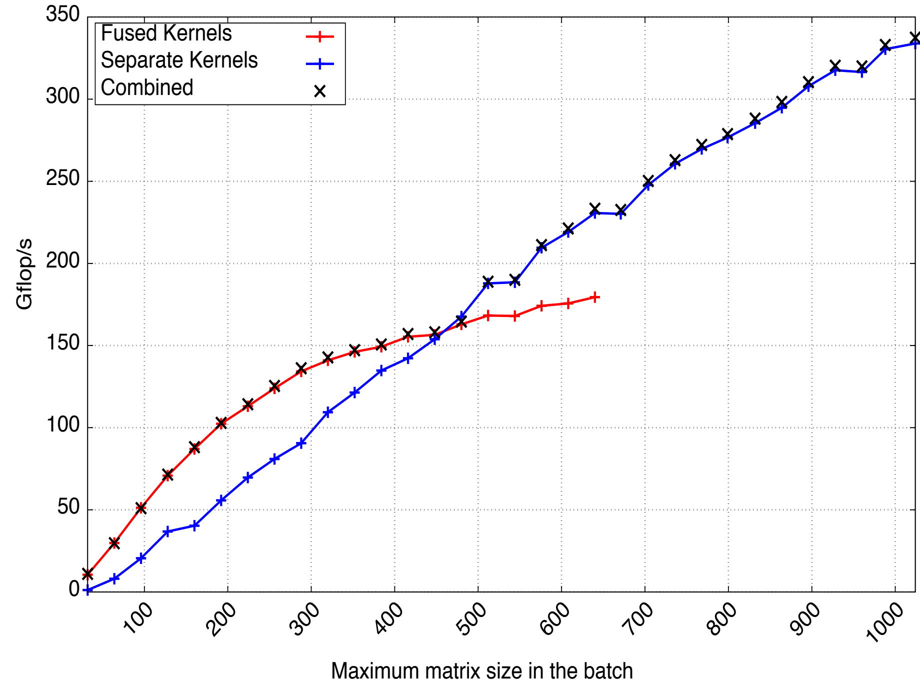


MAGMA Variable size batched Cholesky

DPOTRF on batch of 3000 (Gaussian distribution)



Performance of vbatched fused kernels approach



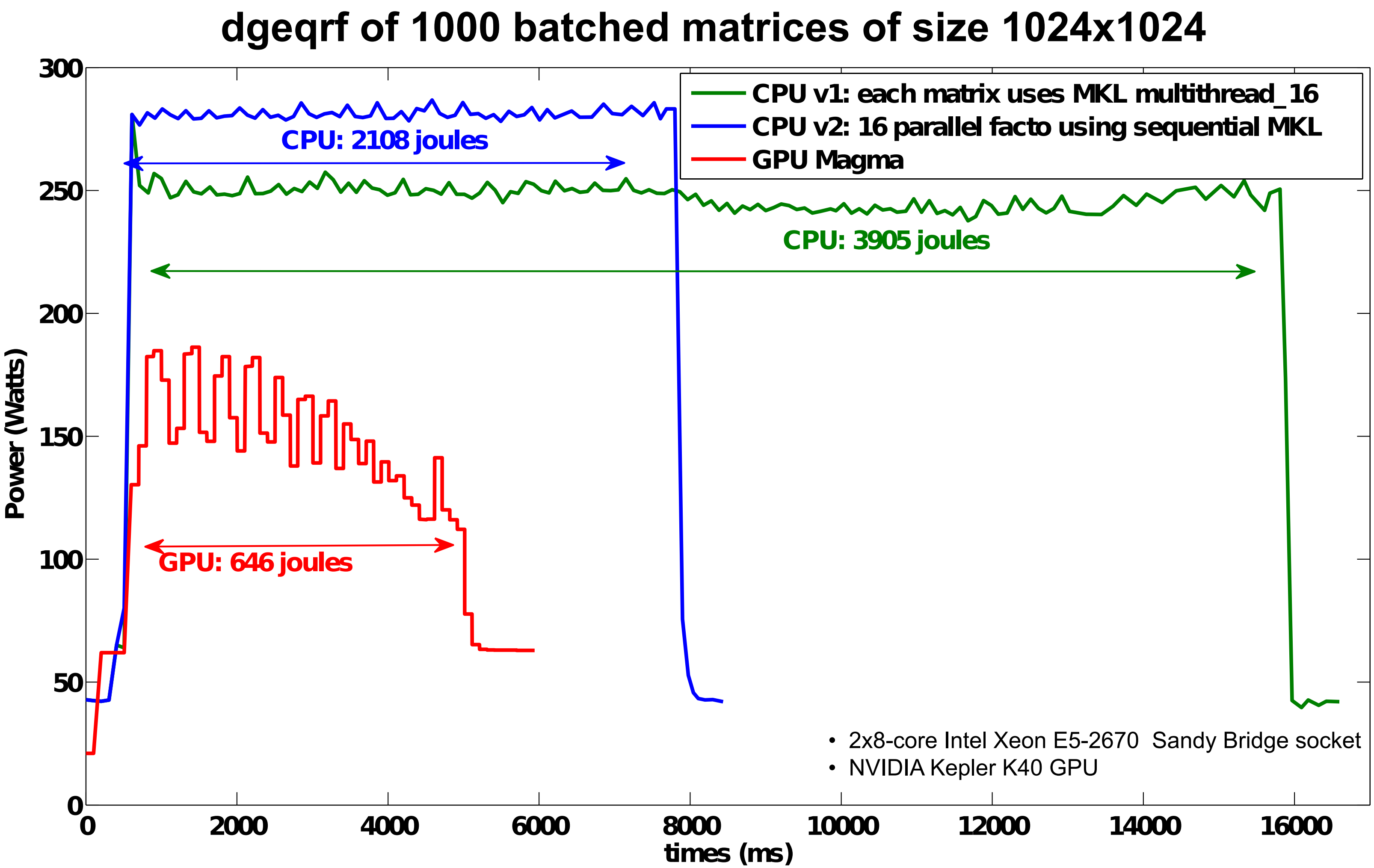
Crossover of fused vs. separate BLAS kernels

- 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

Energy efficiency



CPU does not include DRAM power

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

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

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
University of California, Berkeley
University of Colorado, Denver
INRIA, France (StarPU team)
KAUST, Saudi Arabia