Accelerating Tensor Contractions in High-Order FEM with MAGMA Batched

Ahmad Abdelfattah[1], Marc Baboulin[2], Veselin Dobrev[3], Jack Dongarra[1,4], Chris Earl[3], Joel Falcou[2], Azzam Haidar[1], Ian Karlin[3], Tzanio Kolev[3], Ian Masliah[2], and **Stan Tomov[1]**

[1] Innovative Computing Laboratory, University of Tennessee, Knoxville
[2] University of Paris-Sud, France
[3] Lawrence Livermore National Laboratory, Livermore, CA, USA
[4] University of Manchester, Manchester, UK

SIAM Conference on Computational Science and Engineering (SIAM CSE'17)
Algorithms and Libraries for Tensor Contractions (MS47)
https://www.siam.org/meetings/cse17/
Atlanta, GA, U.S.A.
February 26–March 3, 2017

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Outline

- Introduction
- Tensors in numerical libraries
- Tensor formulation for high-order FEM
- Tensor contractions interfaces and code generation
- Algorithms design and tuning
- Performance
- Conclusions

Introduction

Numerous important applications:

- High-order FEM simulations
- Signal Processing
- Numerical Linear Algebra
- Numerical Analysis
- Data Mining
- Deep Learning
- Graph Analysis
- Neuroscience

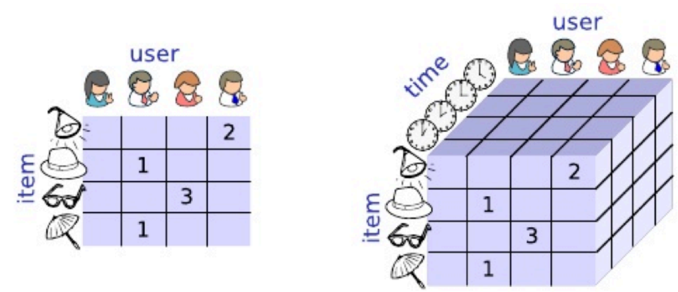
  and more

can be expressed through tensors.

The goal is to design a:

- High-performance package for Tensor algebra;
- Built-in architecture-awareness (GPU, Xeon Phi, multicore);
- User-friendly interface**.**

e.g., relational data



matrix　　　3 order tensor

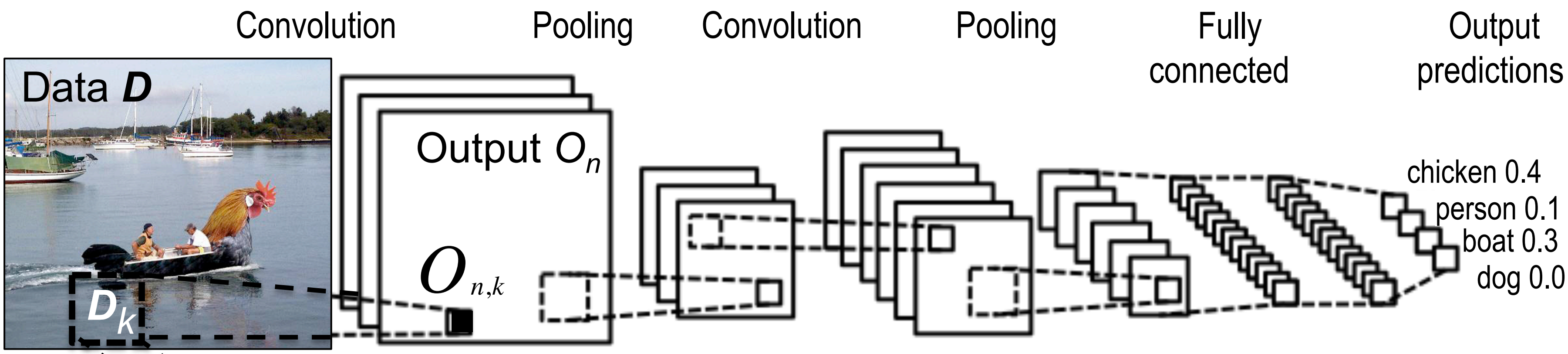
Item	⇔	scalar	tensors	(0)
Items	⇔	vector	tensors	(1)
Relations of pairs	⇔	matrix	tensors	(2)
Relations of 3-tuple	⇔	3-D array	tensors	(3)
…				
Relations of N-tuples	⇔	N-D array	tensors	(N)

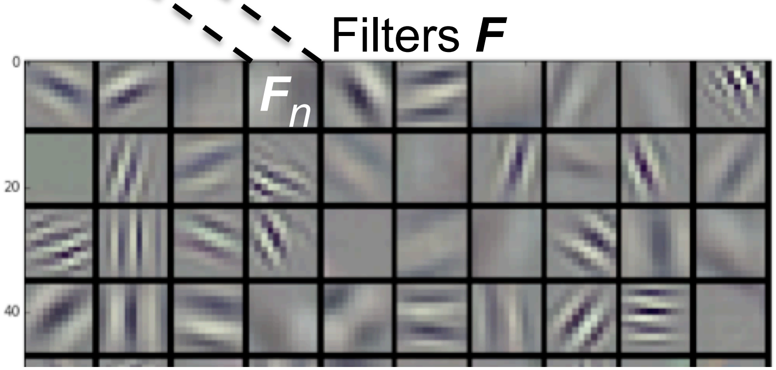
ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

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 of Filters F_i (feature detection) and input image D:

- 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** (for efficiency; among other approaches)

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

Examples

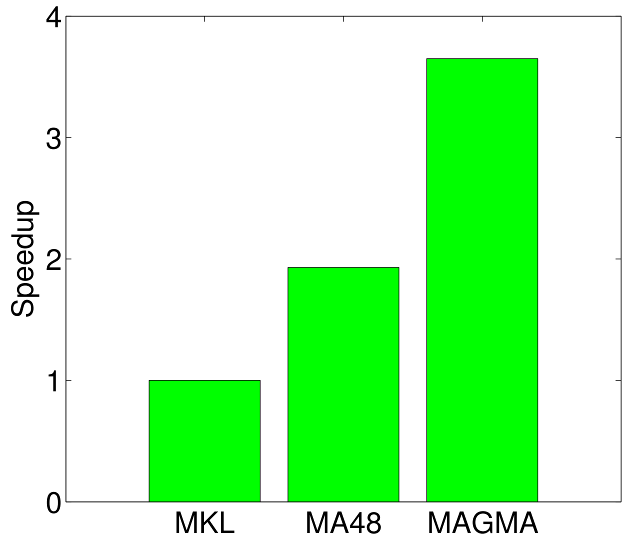
Multi-physics problems need **small & many tensor contractions**

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

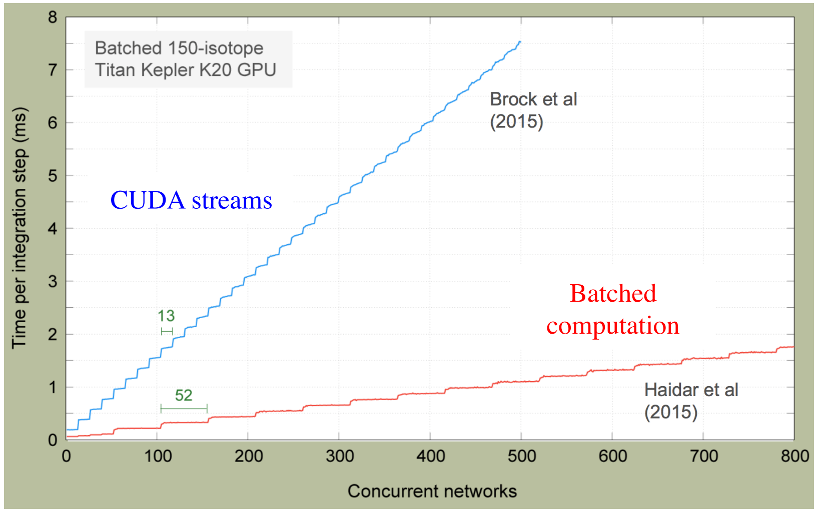
need batched tensor contractions of variable sizes

Speedup of the solver for matrix size 150



3.7x speedup

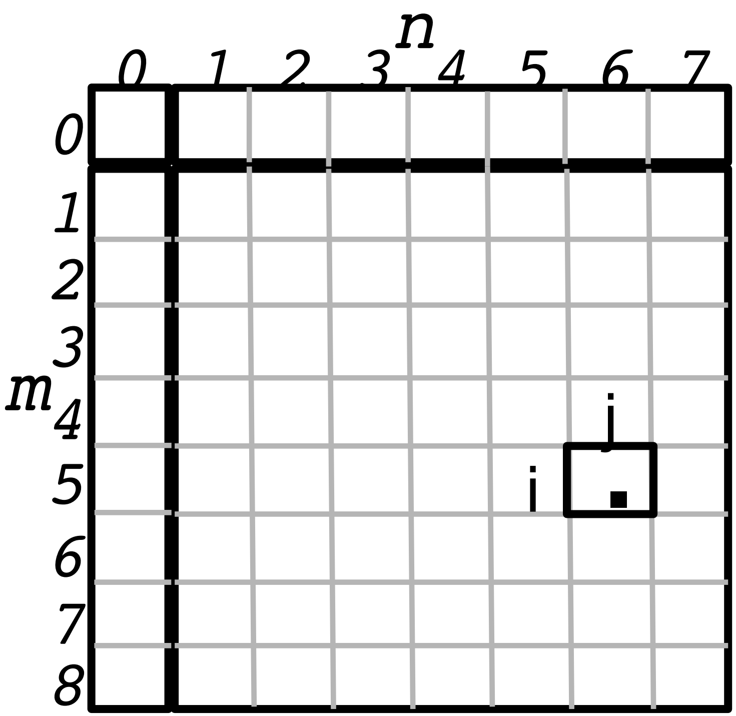
Additional acceleration achieved through MAGMA Batched



7x speedup

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

Tensor abstractions and numerical dense linear algebra



Matrix *A*
In tile data layout

Matrix ***A*** in tiled data-layout as a **4th-order tensor**:

$$A_{i,j,m,n}$$

A rank-64 update as **tensor contraction on index k**:

for i = 0..63

for j = 0..63

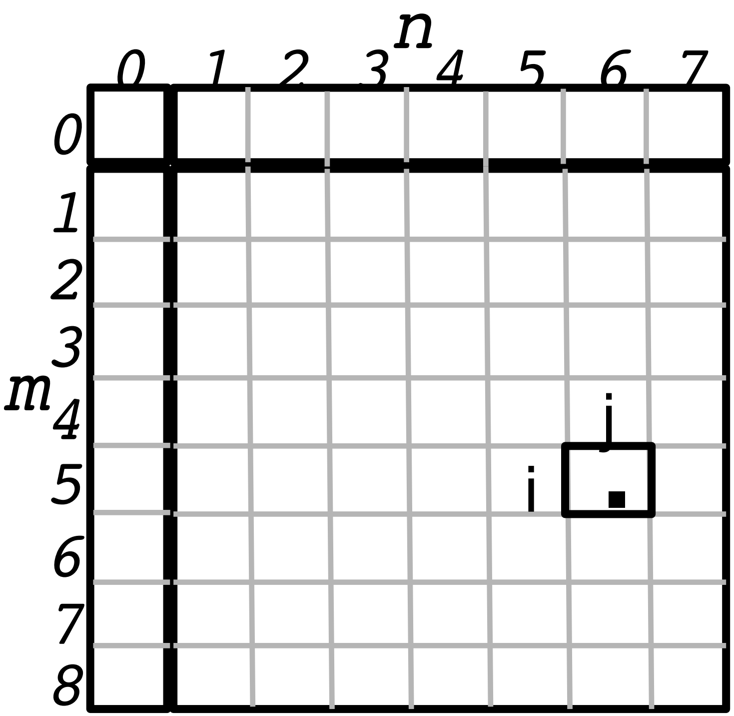
for m = 1..8

for n = 1..7

$$A_{i,j,m,n} -= \sum_k A_{i,k,m,0} A_{k,j,0,n}$$

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

Tensor abstractions and numerical dense linear algebra …



$A_{i,j,m,n}$

How to design it?

```
//Declare a 4th-order Tensor A on the GPU
Tensor<64, 64, 9, 8, gpu_t> A;
```

```
// DSEL design using Einstein notation: repeated
// index k means a summation/contraction.
// Range of the other indices is full/range as
// given through the left assignment operand
A(i, j, m:1..8, n:1..7) -= A(i,k,m,0) * A(k, j,0,n);
```

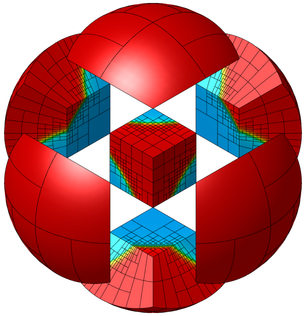
How to implement it?

- Can be casted to BLAS
- Can be very inefficient, e.g., if implemented as dot-products (Level 1 BLAS)
- Better, if
  - Recognized as Level 2 BLAS
  - Recognized as Level 3 BLAS
  - Batched Level 3 BLAS, e.g., GEMM
  - On the fly data reshape
  - …

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

Tensors formulation for high-order FEM

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

Momentum Conservation: $$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{M}_{\mathbf{v}}^{-1}\mathbf{F}\cdot\mathbf{1}$$

Energy Conservation: $$\frac{\mathrm{d}\mathbf{e}}{\mathrm{d}t} = \mathbf{M}_{\mathbf{e}}^{-1}\mathbf{F}^{\mathbf{T}}\cdot\mathbf{v}$$

Equation of Motion: $$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M}_{\mathbf{v}}$ and $\mathbf{M}_{\mathbf{e}}$ 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)

Tensors formulation for high-order FEM

- Consider the FE mass matrix M_E for an element E with weight ρ, as a **2-D tensor**

$$(M_E)_{ij} = \sum_{k=1}^{nq} \alpha_k \, \rho(q_k) \, \varphi_i(q_k) \, \varphi_j(q_k) \, |J_E(q_k)|$$

i*, *j = 1,..., ***nd*** , where

- nd is the number of FE degrees of freedom (dofs)
- nq is the number of quadrature points
- $\{\varphi_i\}_{i=1}^{nd}$ are the FE basis functions on the reference element
- $|J_E|$ is the determinant of the element transformation
- $\{q_k\}_{k=1}^{nq}$ and $\{\alpha_k\}_{k=1}^{nq}$ are the points and weights of the quadr

- Take the ***nq*** x ***nd*** matrix $B_{ki} = \varphi_i(q_k)$, and $(D_E)_{kk} = \alpha_k \, \rho(q_k) \, |J_E(q_k)|$.
  Then, $(M_E)_{ij} = \sum_{k=1}^{nq} B_{ki} (D_E)_{kk} B_{kj}$, or omitting the ***E*** subscript
  $M = B^T D B$.

- Using FE of order *p*, we have $nd = O(p^d)$ and $nq = O(p^d)$, so B is dense $O(p^d)$ x $O(p^d)$ matrix.

- If the FE basis and the quadrature rule have tensor product structure, we can decompose dofs and quadrature point indices in logical coordinate axes
  $i = (i_1, ..., i_d)$, $j = (j_1, ..., j_d)$, $k = (k_1, ..., k_d)$
  so in 3D for example (d=3), M_{ij} can be viewed as 6-dimensional tensor

$$M_{i_1,i_2,i_3,j_1,j_2,j_3} = \sum_{k_1,k_2,k_3} (B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1})(B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2})(B^{1d}_{k_3,i_3} B^{1d}_{k_3,j_3}) D_{k_1,k_2,k_3}$$

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), San Diego, CA, June 6—8, 2016.

Tensor kernels for assembly/evaluation

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)

Index reordering/reshape

If we store tensors as column-wise 1D arrays,

$$M^{nd_1\times nd_2\times nd_1\times nd_2}_{i_1,i_2,j_1,j_2} = M^{nd\times nd}_{i,j} = M^{nd^2}_{i+nd\,j} = M^{nd^2}_{i_1+nd_1 i_2+nd(j_1+nd_1 j_2)}$$

, *i.e.*, $\boldsymbol{M}$ can be interpreted as a 4th order tensor, a $\boldsymbol{nd}$ x $\boldsymbol{nd}$ matrix, or a vector of size $\boldsymbol{nd}^2$, without changing the storage. We can define

$$Reshape(T)^{m_1\times\cdots\times m_q}_{j_1,\cdots,j_q} = T^{n_1\times\cdots\times n_r}_{i_1,\cdots,i_r}$$

as long as $n_1 \ldots n_r = m_1 \ldots m_q$ and for every

$i_{1..r}, j_{1..q}\ i_1 + n_1 i_2 + \ldots + n_1 n_2 \ldots n_{r-1} i_r = j_1 + m_1 j_2 + \ldots + m_1 m_2 \ldots m_{q-1} j_q$.

Contractions can be implemented as a sequence of pairwise contractions. There is enough complexity here to search for something better: code generation, index reordering, and autotuning will be used, *e.g.*, contractions (3a) - (4f) can be implemented as tensor index-reordering plus gemm $\boldsymbol{A}, \boldsymbol{B} \rightarrow \boldsymbol{A}^T\boldsymbol{B}$.

For example:

$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

Can be written as

$Reshape(C)^{nd1\times(nd2nd3)} =$

$A^T\ Reshape(B)^{nq1\times(nd2nd3)}$

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

Tensor contraction interfaces and code generation

- **Design**
  - **Convenience of use** (dimension and data layout abstraction)
  - **Readability** (considered DSEL; decided C++14 is expressive enough)
  - **Performance** (reshape to GEMMs, design, autotuning, compiler – code gen/templates)
- Use **C++14** standard and in particular **constexpr** specifier
  (to evaluate value of function or variable at compile time)

```
// Template specialization
constexpr auto layout = of_size<5,3>();
// Using Integral constant
constexpr auto layout1 = of_size(5_c,3_c);
// Using dynamic dimensions
constexpr auto layout2 = of_size(5,3);
// Access Dimensions at compile time
constexpr auto dim1 = layout(1);
```

Listing 1: Dimensions for Tensors

```
// Create a rank 2 tensor of size 5,3 on GPU
constexpr tensor<float,gpu_> d_ts(of_size<5,3>());
// Create a rank 2 tensor of size 5,3 on CPU
constexpr tensor<float> ts(of_size<5,3>());
// Use thrust to fill d_ts with 9
thrust::fill(d_ts.begin() , d_ts.end() , 9);
// Copy d_ts from GPU to ts on CPU
copy( d_ts , ts ) ;
```

Listing 2: Create Tensor and copy

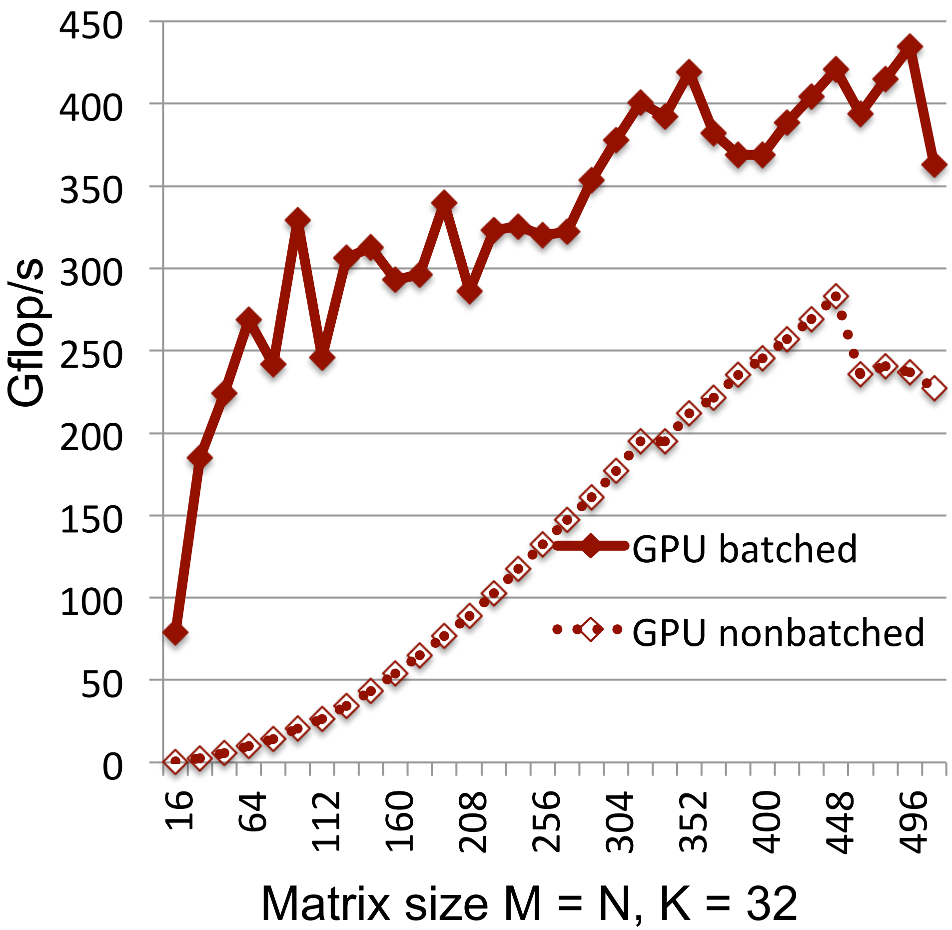
```
// Create a batch that will contain 15 tensors of size 5,3,6
constexpr auto batch<float, gpu_> b = make_batch(of_size(5_c,3_c,6_c) , 15);
// Accessing a tensor from the batch returns a view on it
constexpr auto view_b = b(0);
// Create a grouping of tensors of same size tensors
constexpr auto group<float,gpu_> g(of_size(5_c,3_c));
// Add a tensor to the group
constexpr auto tensor<float,gpu_> d_ts( of_size(5_c,3_c) );
g.push_back(d_ts);
```

Listing 3: Batched tensors

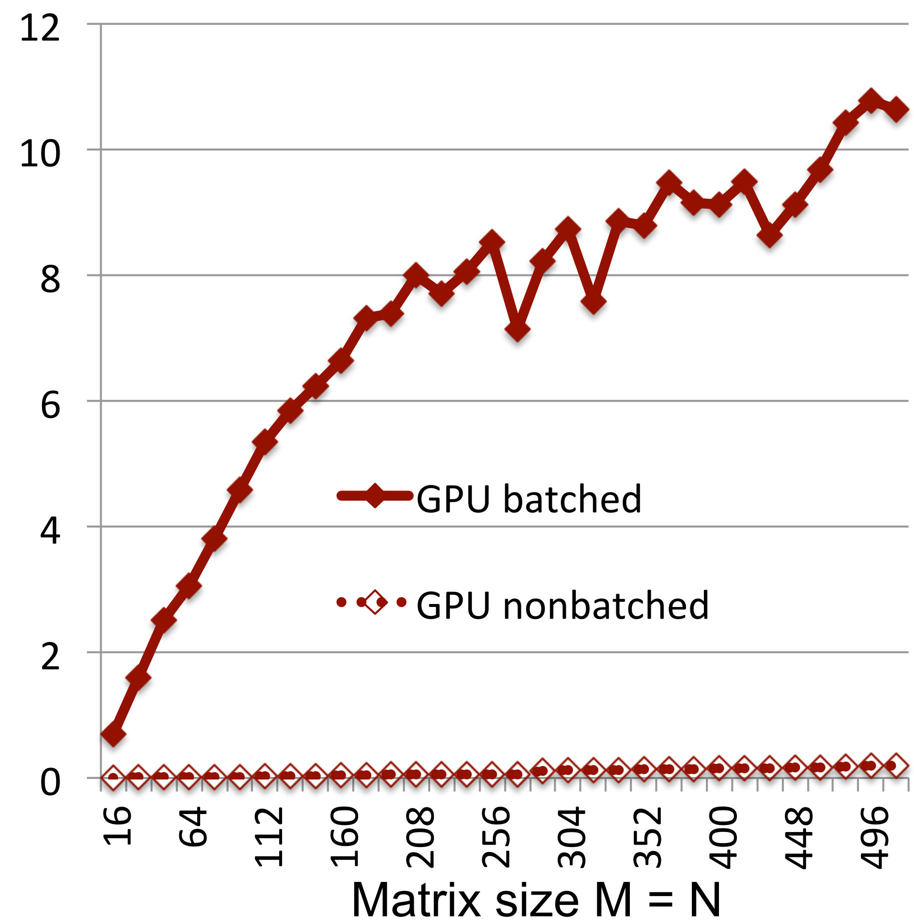
Algorithm designs

- Importance of **reshaping to GEMMs**: as illustrated, **not all flops are equal**

DGEMM (NN), batch_count = 500, 1 Tesla K40c GPU



DAXPY, batch_count = 500, 1 Tesla K40c GPU



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

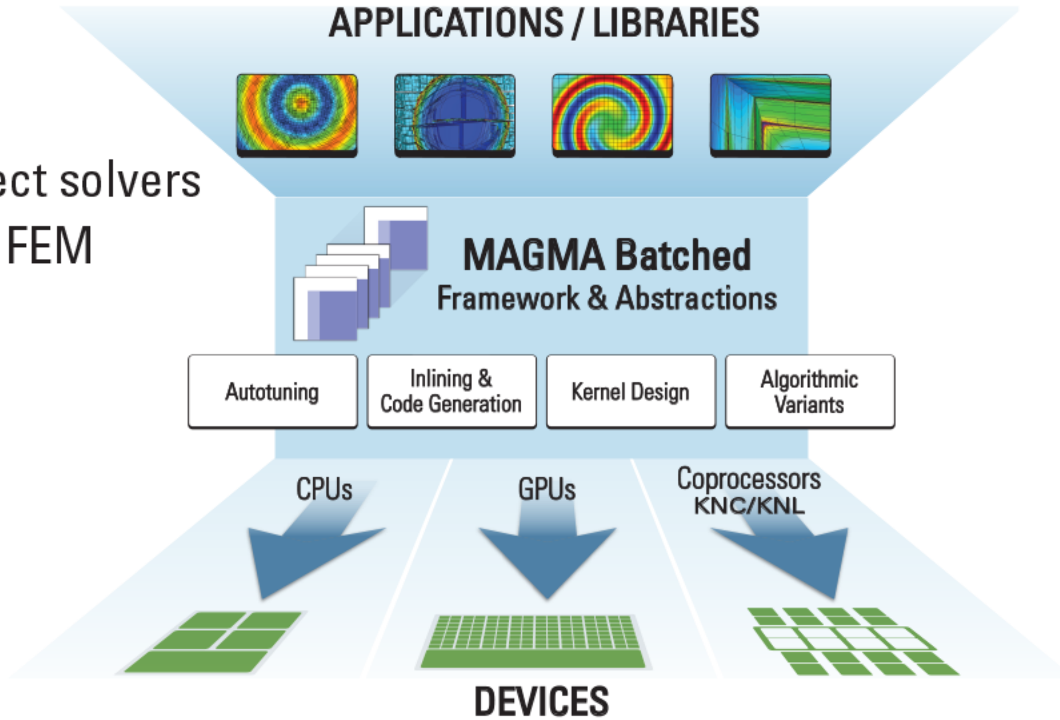
Batched routines released in MAGMA

MAGMA BATCHED

BATCHED FACTORIZATION OF A SET OF SMALL MATRICES IN PARALLEL

Numerous applications require factorization of many small matrices

- Deep learning
- Structural mechanics
- Astrophysics
- Sparse direct solvers
- High-order FEM simulations



ROUTINES

LU, QR, and Cholesky	✓
Solvers and matrix inversion	✓
All BLAS 3 (fixed + variable)	✓
SYMV, GEMV (fixed + variable)	✓

Implementation on current hardware is becoming challenging

Memory hierarchies

Memory hierarchies	Haswell E5-2650 v3	KNL 7250 DDR5|MCDRAM	ARM	K40c	P100
	10 cores	68 cores	4 cores	15 SM x 192 cores	56 SM x 64 cores
Registers	16/core AVX2	32/core AVX-512	32/core	256 KB/SM	256 KB/SM
L1 cache & GPU shared memory	32 KB/core	32 KB/core	32 KB/core	64 KB/SM	64 KB/SM
L2 cache	256 KB/core	1024 KB/2cores	2 MB	1.5 MB	4 MB
L3 cache	25 MB	0...16 GB	N/A	N/A	N/A
Main memory	64 GB	384 |16 GB	4 GB	12 GB	16 GB
Main memory bandwidth	68 GB/s	115 | 421 GB/s	26 GB/s	288 GB/s	720 GB/s
PCI express gen3 x16	16 GB/s	16 GB/s	16 GB/s	16 GB/s	16 GB/s
Interconnect Cray Gemini	6 GB/s	6 GB/s	6 GB/s	6 GB/s	6 GB/s

Memory hierarchies for different type of architectures

Workshop on Batched, Reproducible, and Reduced Precision BLAS

Georgia Tech
Computational Science and Engineering
Atlanta, GA
February 23—25, 2017

http://bit.ly/Batch-BLAS-2017

Draft Reports
Batched BLAS Draft Reports:
https://www.dropbox.com/s/olocmipyxfvcaui/batched_api_03_30_2016.pdf?dl=0

Batched BLAS Poster:
https://www.dropbox.com/s/ddkym76fapddf5c/Batched%20BLAS%20Poster%2012.pdf?dl=0

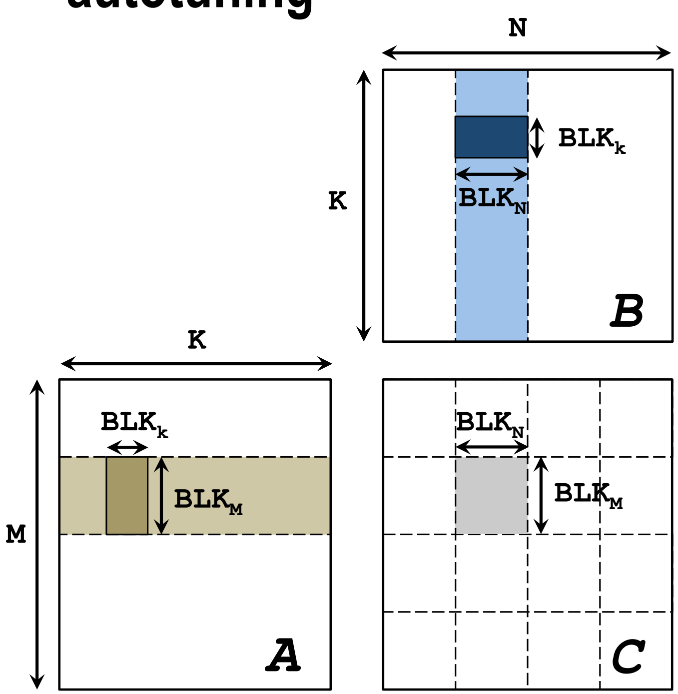
Batched BLAS Slides:
https://www.dropbox.com/s/kz4fhcipz3e56ju/BatchedBLAS-1.pptx?dl=0

Webpage on ReproBLAS:
http://bebop.cs.berkeley.edu/reproblas/

Efficient Reproducible Floating Point Summation and BLAS:
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-229.pdf

Algorithm designs …

- **Reshape to GEMMs**
- **GEMM is multilevel blocked** code from MAGMA to map to GPU's hierarchical memory
- **Parametrized for autotuning**



- **Use Batched execution**
  - In general 1 TB per matrix
  - Use vectorization across matrices in a TB for very small matrices; we denote by **TB Concurrency (tbc)**
- **Templates** and **constexpr** to avoid param. checking and compiler-unrolled code
- No pointers to batched matrices: passed through formulas in the tensor abstraction
- General kernel organization:

1) Read A and B (or parts if blocking) in fast memory
   - through functions in the tensor abstraction for layout
   - allows for **on-the-fly reshape** (data for indices in the operation may not be in standard GEMM form)
2) Compute, e.g., A B
3) Update C

Autotuning

1) Kernel variants: performance parameters are exposed through a templated kernel interface

```
template< typename T,  int    DIM_X,   int    DIM_Y,
                       int    BLK_M,   int   BLK_N,   int    BLK_K,
                       int   DIM_XA,   int  DIM_YA,   int  DIM_XB,   int  DIM_YB,
                       int    THR_M,   int   THR_N,   int    CONJA,   int   CONJB  >
static __device__ void  tensor_template_device_gemm_nn( int  M,  int  N,  int  K, …
```

2) CPU interfaces that call the GPU kernels as a Batched computation

```
template<typename T,  int    DIM_X,  int   DIM_Y, … >
void tensor_template_batched_gemm_nn( int m, int n, int k, … ) {
     …
   tensor_template_device_gemm _nn<T, DIM_X, DIM_Y, … ><<<dimGrid, dimBlock, 0, queue>>>(m, n, k,…);
}
```

3) Python scripts that generate the search space for the parameters DIM_X, DIM_Y …

```
                     index,     DIM_X,   DIM_Y,  …
#define NN_V_0           4,          8,    8,   24,    8,    1,    4,    8,    4,    8
#define NN_V_1           4,          8,    8,   32,    8,    1,    4,    8,    4,    8
#define NN_V_2           4,          8,    8,   40,    8,    1,    4,    8,    4,    8
…
```

4) Scripts that run all versions in the search space, analyze the results, and return the best combination of parameters, which is stored in the library for subsequent use.

Performance model

$$P_{max} = \frac{F}{T_{min}}$$

Flops for the computation (F)

Fastest time to solution (T_{min})

- For square matrices

  $F \approx 2n^3$, $\quad T_{min} = min_T\,(T_{Read(A,B,C)} + T_{Compute(C)} + T_{Write(C)})$

- Need to read/write $4\,n^2$ elements, i.e., $32n^2$ Bytes in DP
  => if max bandwidth is **B**, we can take $T_{min} = 32\,n^2 / B$ in DP. Thus,

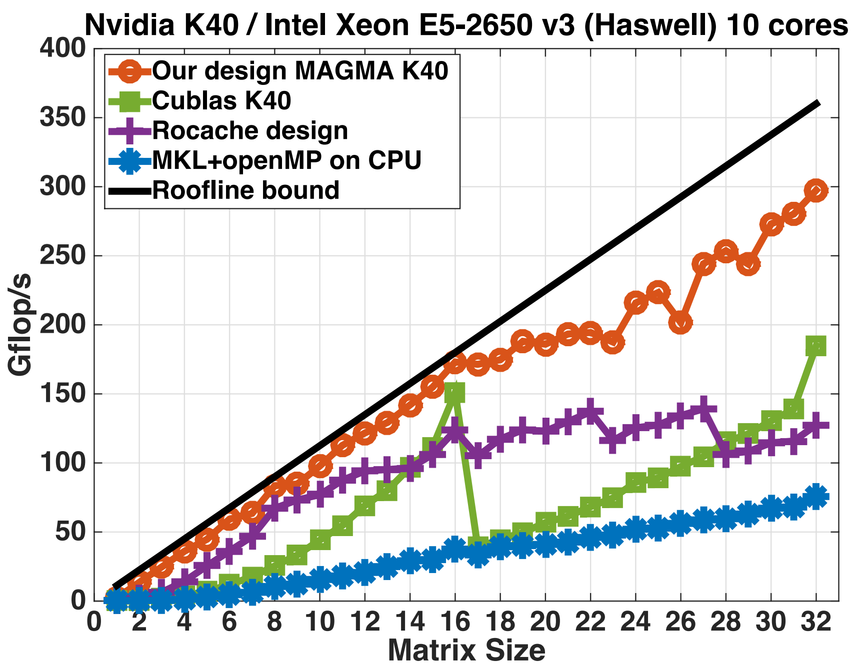
$$P_{max} = \frac{2n^3 B}{32n^2} = \frac{nB}{16} \text{ in DP.}$$

- With ECC on, peak on B on a K40c is ≈180 GB/s, so when n=16 for example, we expect theoretical max performance of 180 Gflop/s in DP

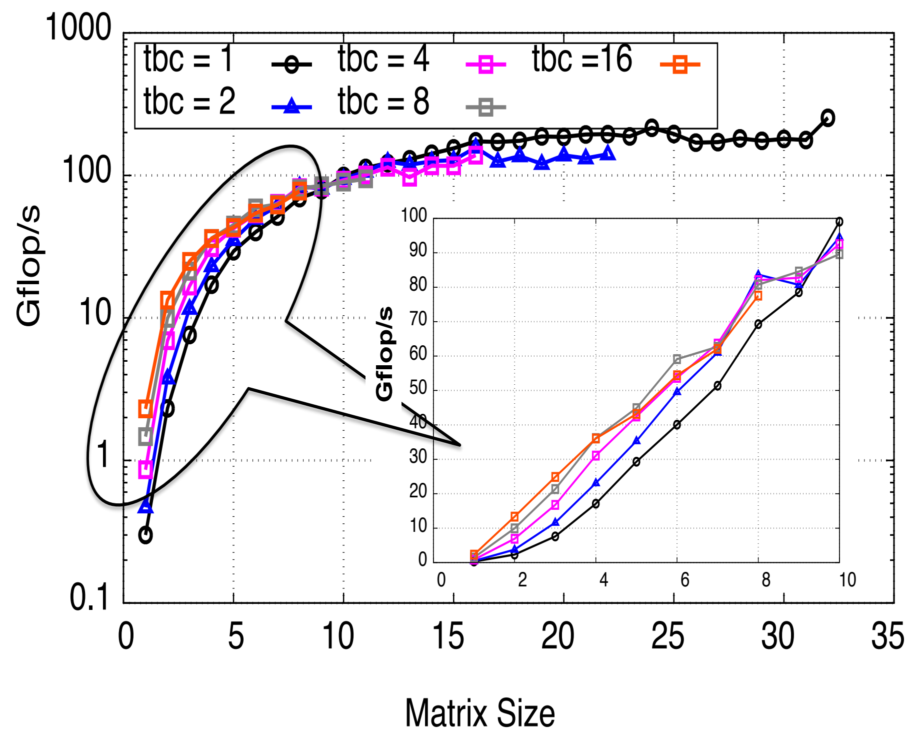
ICL UT INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE Department of Electrical Engineering and Computer Science

Performance results

Performance comparison of tensor contraction versions using batched C = αAB + βC on 100,000 square matrices of size n on a **K40c GPU** and 16 cores of Intel Xeon E5-2670, 2.60 GHz CPUs.

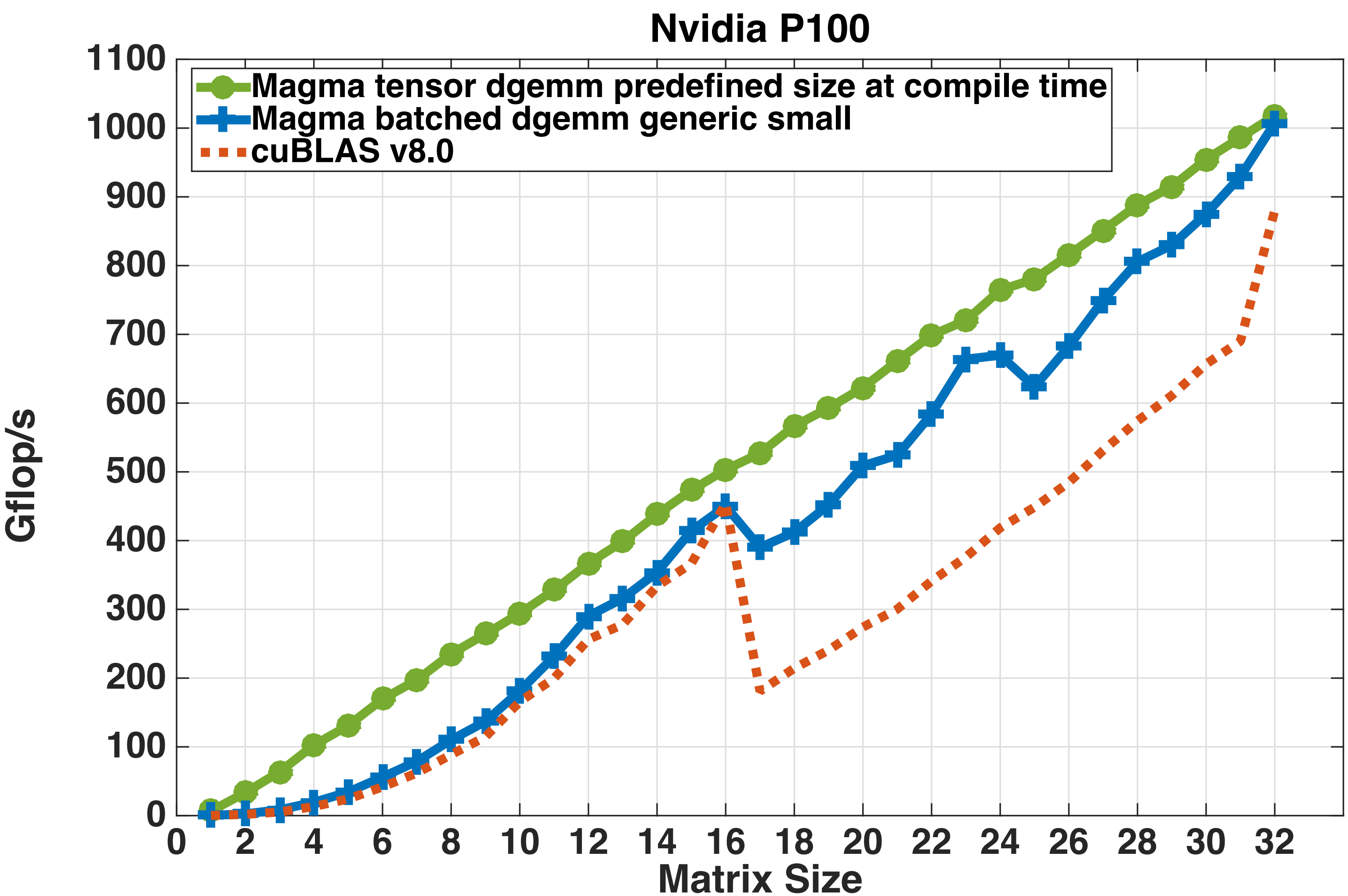


Effect of a Thread Block Concurrency (tbc) techniques where several DGEMMs are performed on one TB simultaneously



THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Performance results

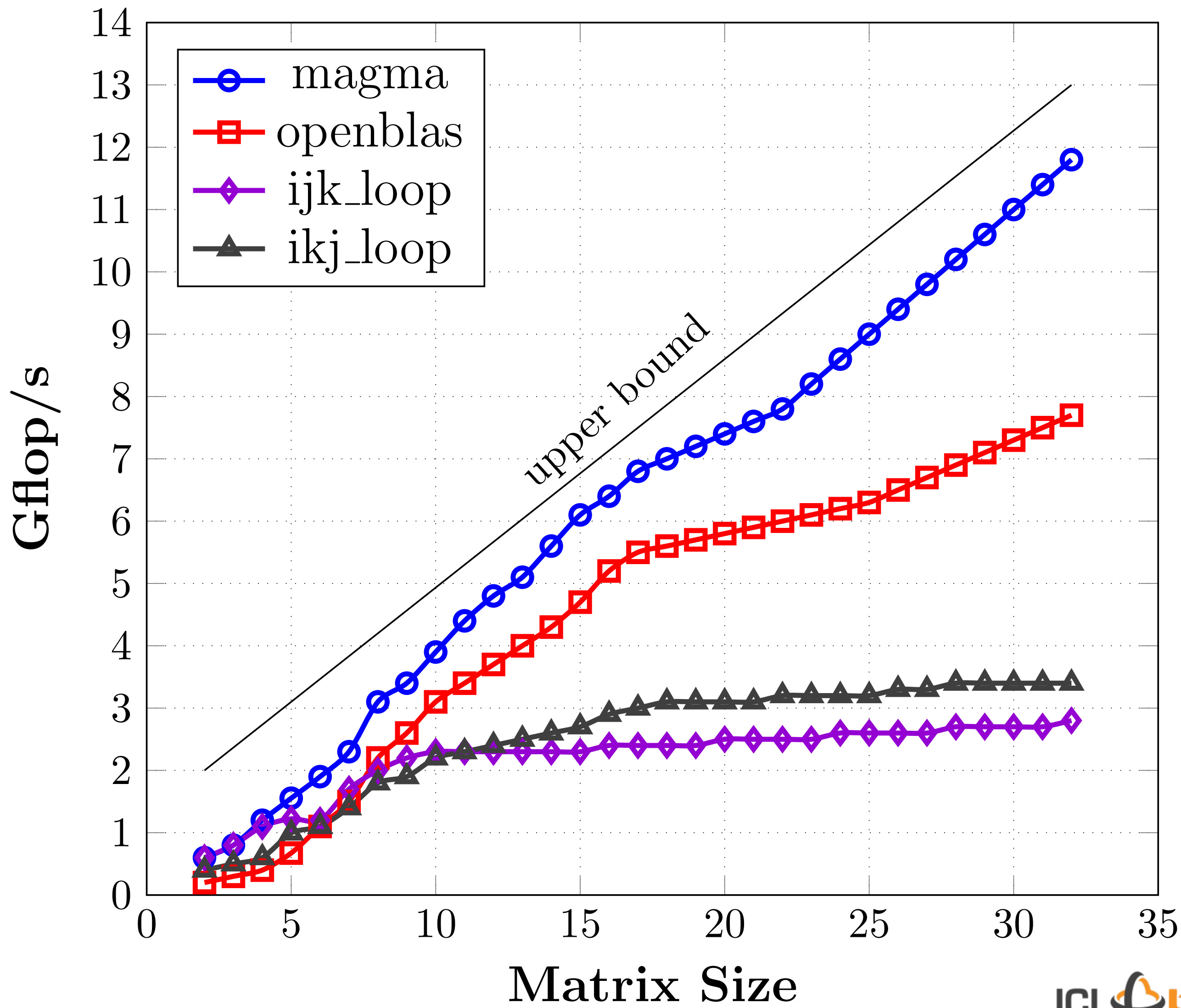


ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Performance results

Batched DGEMM on Tegra ARM



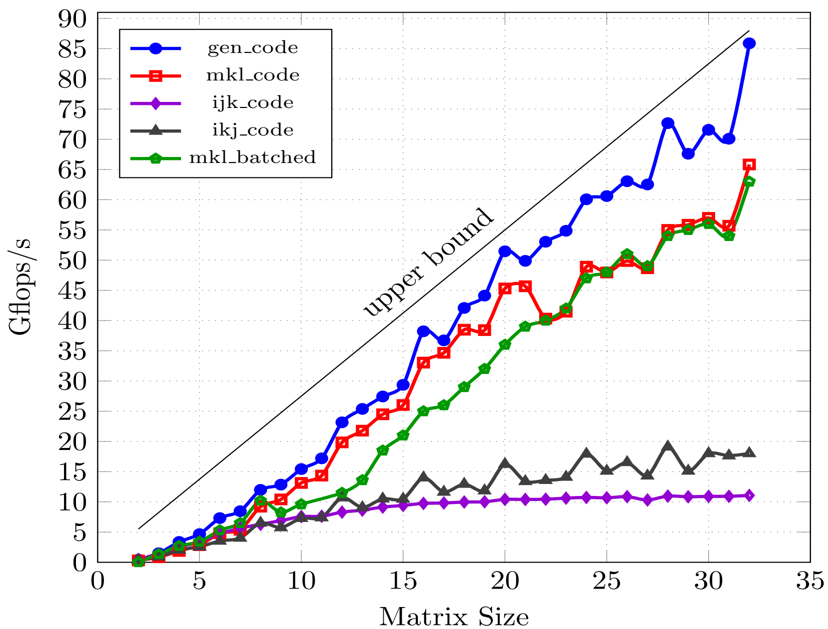
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

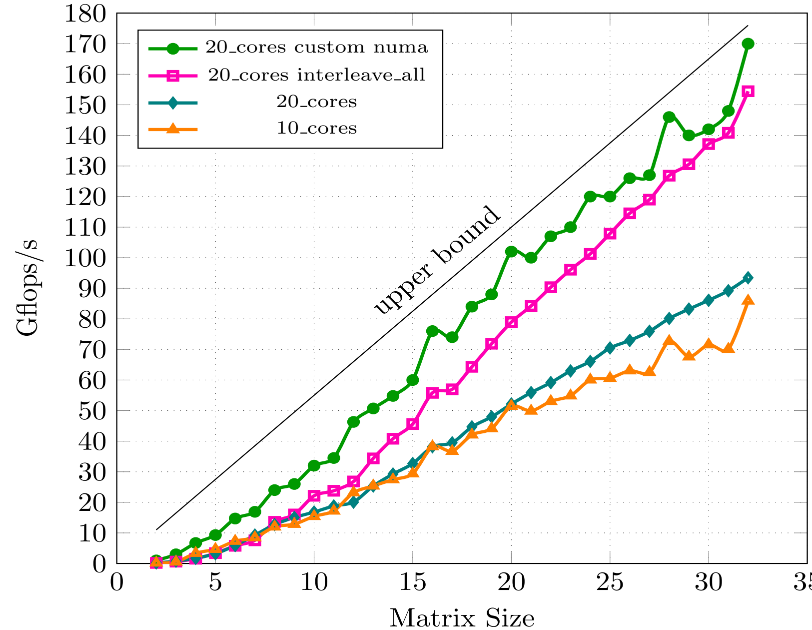
Performance …

Batched DGEMM on CPUs

Intel Xeon E5-2650 v3 (Haswell), 10 cores



2 x Intel Xeon E5-2650 v3 (Haswell), 20 cores

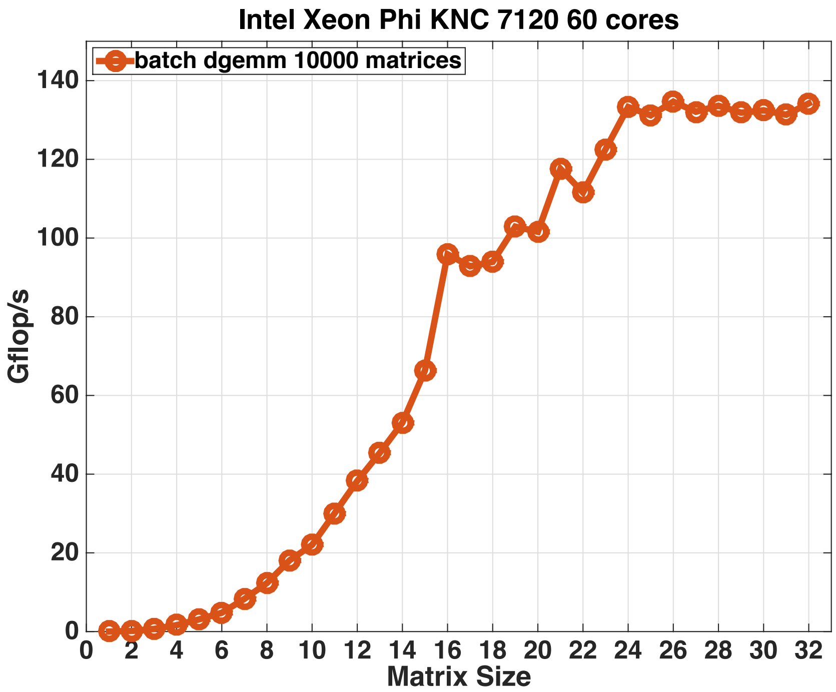


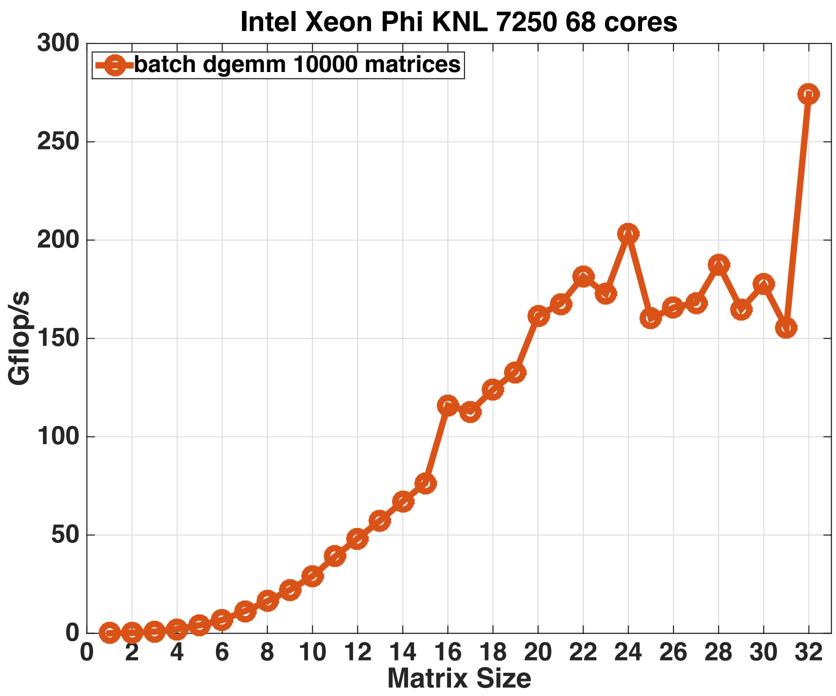
I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, and J. Dongarra, ***High-performance matrix-matrix multiplications of very small matrices***, Euro-Par'16, Grenoble, France, August 22-26, 2016.

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

Performance results

Batched DGEMM on Intel Xeon Phi





ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Conclusions and future work

In conclusion:

- Developed **tensor abstractions** for high-order FEM
- Multidisciplinary effort
- Achieve **90+% of theoretical maximum** on GPUs and multicore CPUs
- Use **on-the-fly tensor reshaping** to cast tensor contractions as **small but many GEMMs**, executed using batched approaches
- Custom designed GEMM kernels for small matrices and autotuning

Future directions:

- To release a tensor contractions package through the MAGMA library
- Integrate developments in BLAST
- Complete autotuning and develop all kernels

ICL 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 Manchester, Manchester, UK
University of Paris-Sud, France
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