GPU TECHNOLOGY CONFERENCE

S7728 - MAGMA Tensors and Batched Computing for Accelerating Applications on GPUs

Stan Tomov - Research Director, UTK
Azzam Haidar - Research Scientist, UTK

Abstract: Learn how to accelerate your machine learning, data mining, and other algorithms through fast matrix and tensor operations on GPUs. There's an increasing demand for accelerated independent computations on tensors and many small matrices. Although common, these workloads cannot be efficiently executed using standard linear algebra libraries. To fill the gap, we developed the MAGMA Batched library that achieves dramatically better performance by repetitively executing the small operations in "batches." We'll describe a methodology on how to develop high-performance BLAS, SVD, factorizations, and solvers for both large- and small-batched matrices. We'll also present the current state-of-the-art implementations and community efforts to standardize an API that extends BLAS for Batched computations.

GTC 2017
San Jose, CA
May 8—11, 2017

CEED
EXASCALE DISCRETIZATIONS

ICL UT
INNOVATIVE
COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering
and Computer Science

GPU TECHNOLOGY CONFERENCE

MAGMA Tensors and Batched Computing for Accelerating Applications on GPUs

Stan Tomov and **Azzam Haidar**
Innovative Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville

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

GTC 2017
San Jose, CA
May 8—11, 2017

CEED
EXASCALE DISCRETIZATIONS

ICL INNOVATIVE COMPUTING LABORATORY

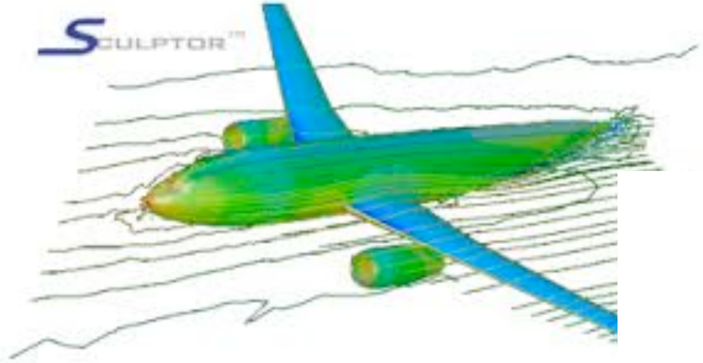
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

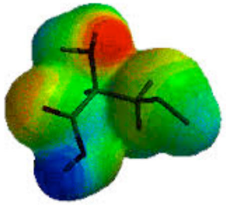
Outline

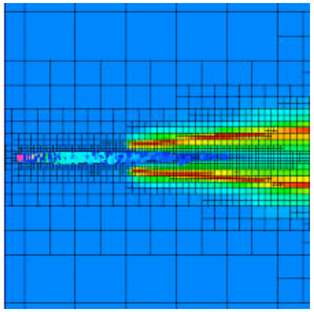
- Introduction
- MAGMA library
  - Numerical Linear Algebra (NLA) for large problems
  - NLA for applications that need small problems
- MAGMA Tensor contraction computations
- MAGMA Batched Computing
- MAGMA-DNN NLA backend for DNN
- Algorithms and optimization techniques
- Conclusions

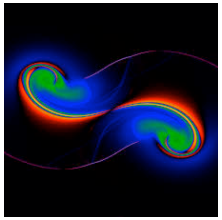
Wide range of Applications depend on Numerical Linear Algebra (NLA) Libraries

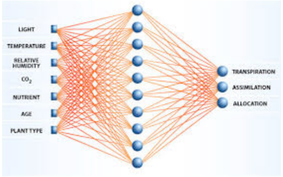
- Airplane wing design,
- Quantum chemistry,
- Geophysical flows,
- Stealth aircraft,
- Diffusion of solid bodies in a liquid,
- Adaptive mesh refinement,
- Computational materials research,
- Deep learning in neural networks,
- Stochastic simulation,
- Massively parallel data mining,
- …

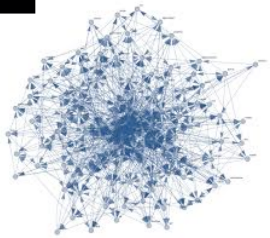












Numerical Linear Algebra (NLA) in Applications

NLA is the backend that accelerates a wide variety of science and engineering applications:

- **Linear system** **Solve $Ax = b$**
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** **Find x to minimize $|| Ax - b ||$**
  - Convex **o**ptimization, Computational statistics (e.g., linear least squares or ordinary least squares), econometrics, control theory, signal processing, curve fitting, and many more
- **Eigenproblems:** **Solve $Ax = \lambda x$**
  - Computational chemistry, quantum mechanics, material science, face recognition, PCA, data-mining, marketing, Google Page Rank, spectral clustering, vibrational analysis, compression, and many more
- **Singular Value Decomposition (SVD):** **$A = U \Sigma V^*$**
  - Information retrieval, web search, signal processing, big data analytics, low rank matrix approximation, total least squares minimization, pseudo-inverse, and many more
- **Many variations depending on structure of A**
  - A can be symmetric, positive definite, tridiagonal, Hessenberg, banded, sparse with dense blocks, etc.
- **LA is crucial to the development of sparse solvers**

Numerical Linear Algebra (NLA) in Applications

NLA is the backend that accelerates a wide variety of science and engineering applications:

- **For big NLA problems**
  (BLAS, convolutions, SVD, linear system solvers, etc.)

Large matrices

In contemporary libraries:
BLAS
LAPACK
ScaLAPACK

MAGMA (for GPUs)

Numerical Linear Algebra (NLA) in Applications

NLA is the backend that accelerates a wide variety of science and engineering applications:

- **For big NLA problems**
  (BLAS, convolutions, SVD, linear system solvers, etc.)

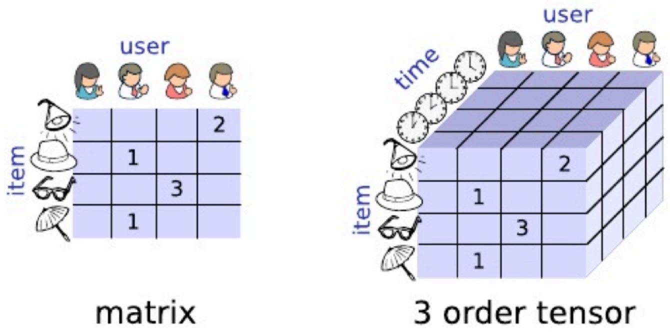
Large matrices

In contemporary libraries:
BLAS
LAPACK
ScaLAPACK

MAGMA (for GPUs)

- **Numerous important applications need NLA for small problems**
  - **Machine learning / DNNs**
  - **Data mining / analytics**
  - High-order FEM,
  - Graph analysis,
  - Neuroscience,
  - Astrophysics,
  - Quantum chemistry,
  - Signal processing, and more

Where data can be multidimensional / relational



Numerical Linear Algebra (NLA) in Applications

NLA is the backend that accelerates a wide variety of science and engineering applications:

- **For big NLA problems**
  (BLAS, convolutions, SVD, linear system solvers, etc.)

Large matrices

In contemporary libraries:
BLAS
LAPACK
ScaLAPACK

MAGMA (for GPUs)

- **Adding in MAGMA application backends for small problems**
  - **Machine learning / DNNs**
  - **Data mining / analytics**
  - High-order FEM,
  - Graph analysis,
  - Neuroscience,
  - Astrophysics,
  - Quantum chemistry,
  - Signal processing, and more

Small matrices / tensors

Fixed-size batches

Variable-size batches

Dynamic batches

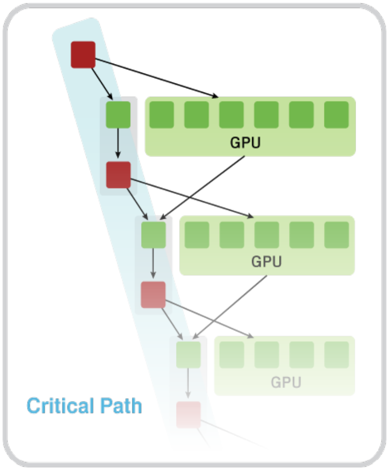
Tensors

Key Features of MAGMA 2.2

TASK-BASED ALGORITHMS

MAGMA uses task-based algorithms where the computation is split into tasks of varying granularity and their execution scheduled over the hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level 3 BLAS, are scheduled on the GPUs.

BLAS tasking + hybrid scheduling



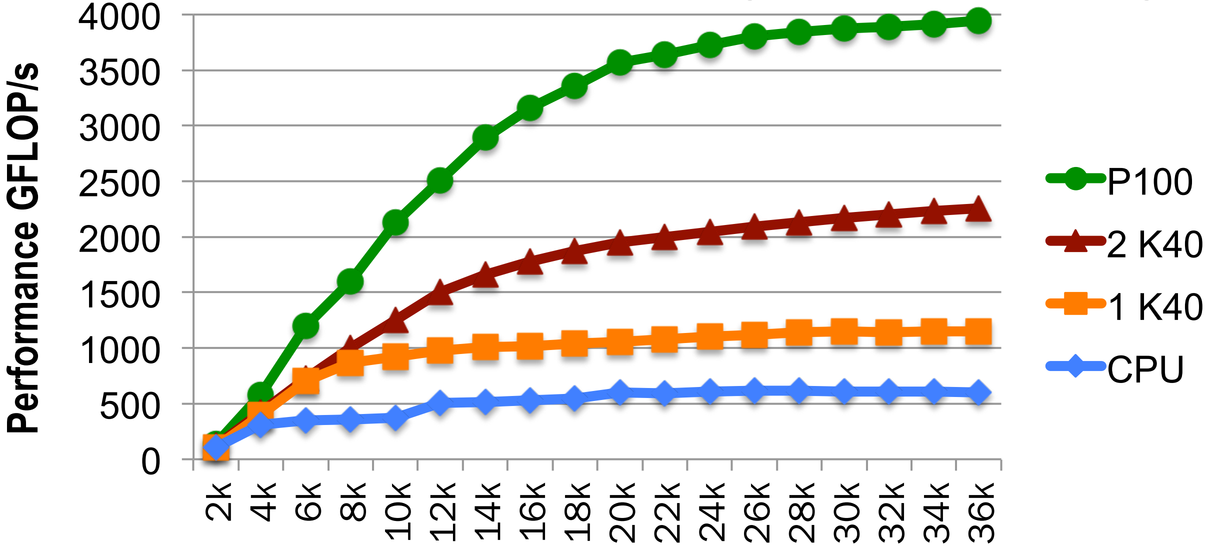
PERFORMANCE & ENERGY EFFICIENCY

MAGMA LU factorization in double precision arithmetic

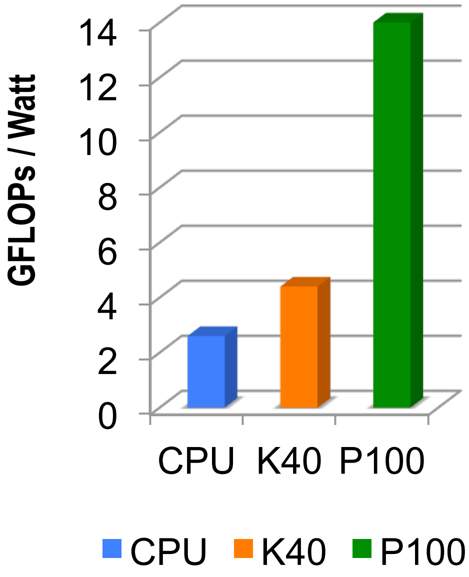
CPU Intel Xeon E5-2650 v3 (Haswell)
2x10 cores @ 2.30 GHz

K40 NVIDIA K40 GPU
15 MP x 192 @ 0.88 GHz

P100 NVIDIA Pascal GPU
56 MP x 64 @ 1.19 GHz



Matrix size N x N



GPU TECHNOLOGY CONFERENCE

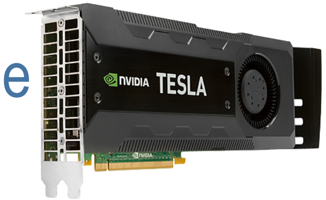
CEED EXASCALE DISCRETIZATIONS

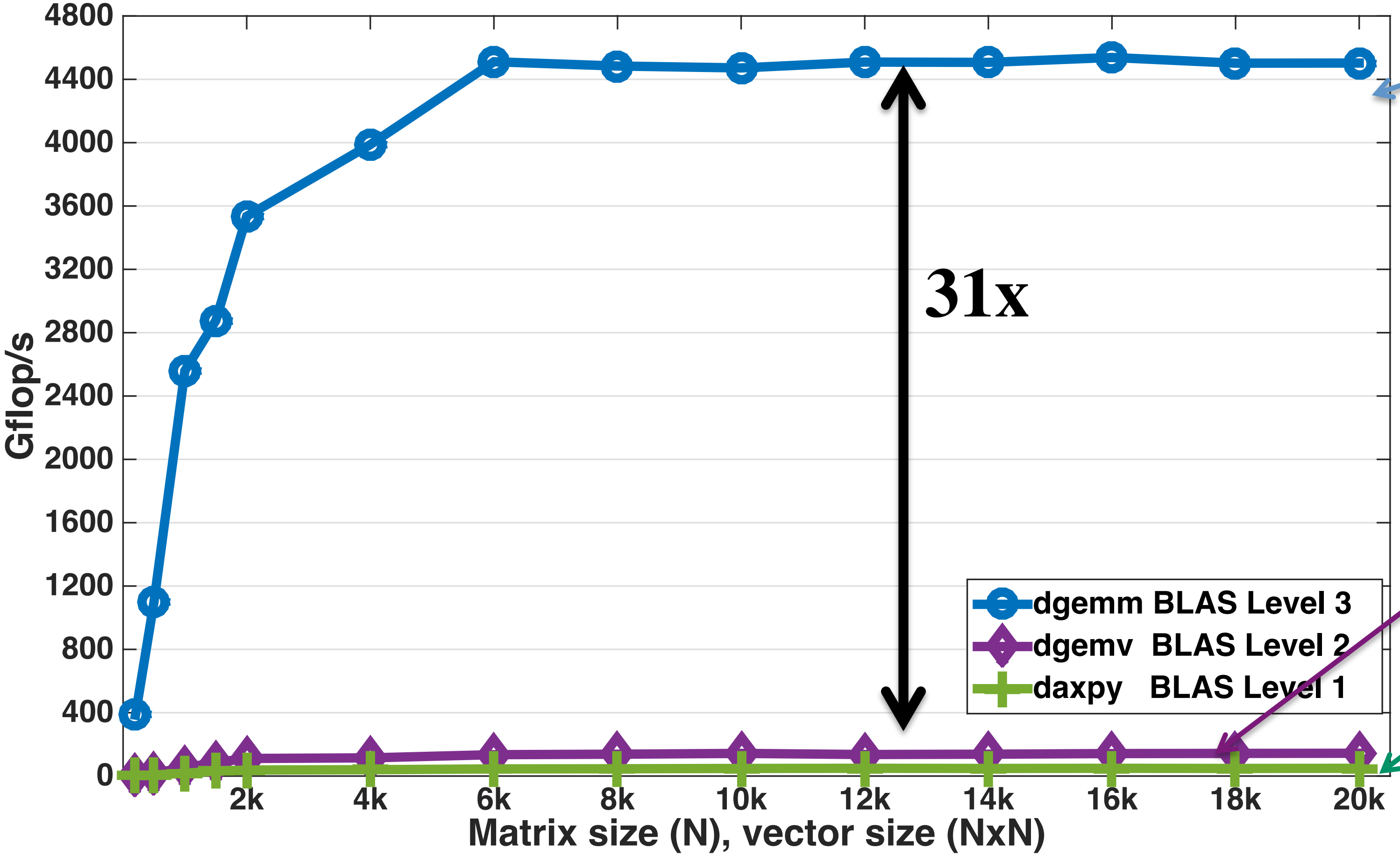
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA – designed to use Level 3 BLAS as much as possible

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 Gflop/s





C = C + A*B

4503 Gflop/s

y = y + A*x

145 Gflop/s

y = α *x + y

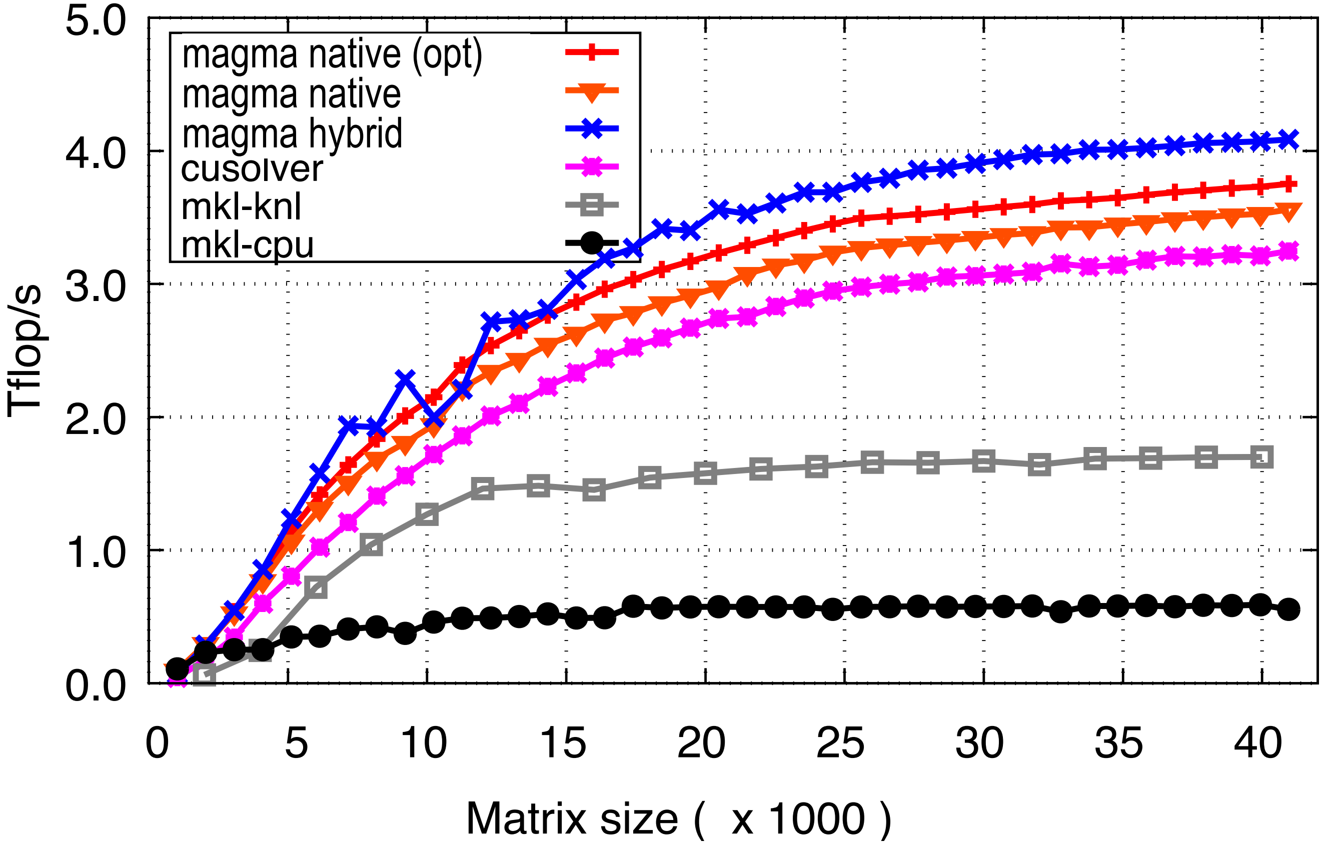
52 Gflop/s

Nvidia P100
The theoretical peak double precision is 4700 Gflop/s
CUDA version 8.0

MAGMA Algorithms (influenced by hardware trend)
Hybrid (using CPU + GPUs) and/vs. GPU-only

MAGMA LU factorization in double precision arithmetic

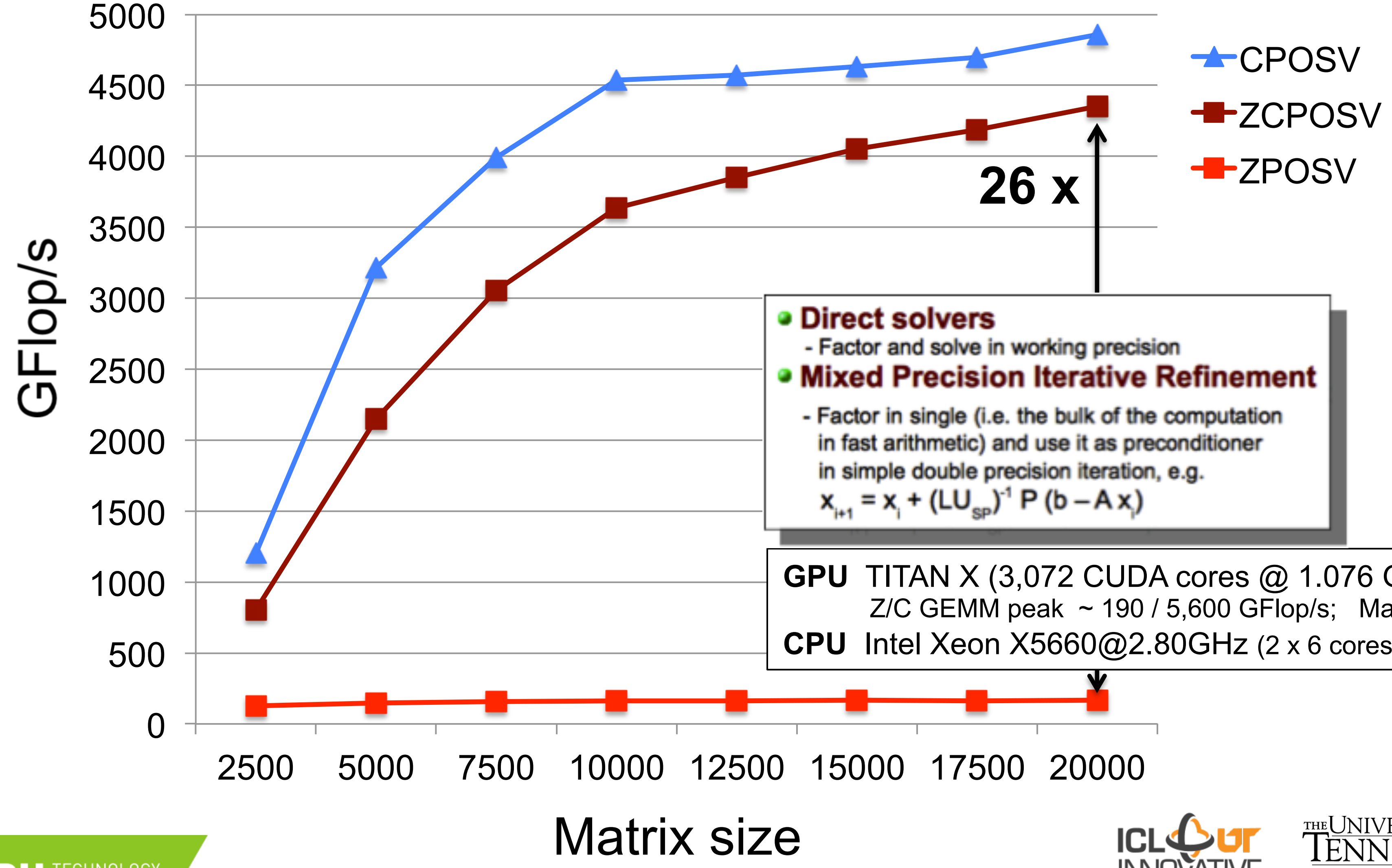
CPU	Intel Xeon E5-2650 v3 (Haswell) 2x10 cores @ 2.30 GHz	**K40**	NVIDIA K40 GPU 15 MP x 192 @ 0.88 GHz	**P100**	NVIDIA Pascal GPU 56 MP x 64 @ 1.19 GHz



MAGMA Algorithms (influenced by hardware trend)

Mixed-precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



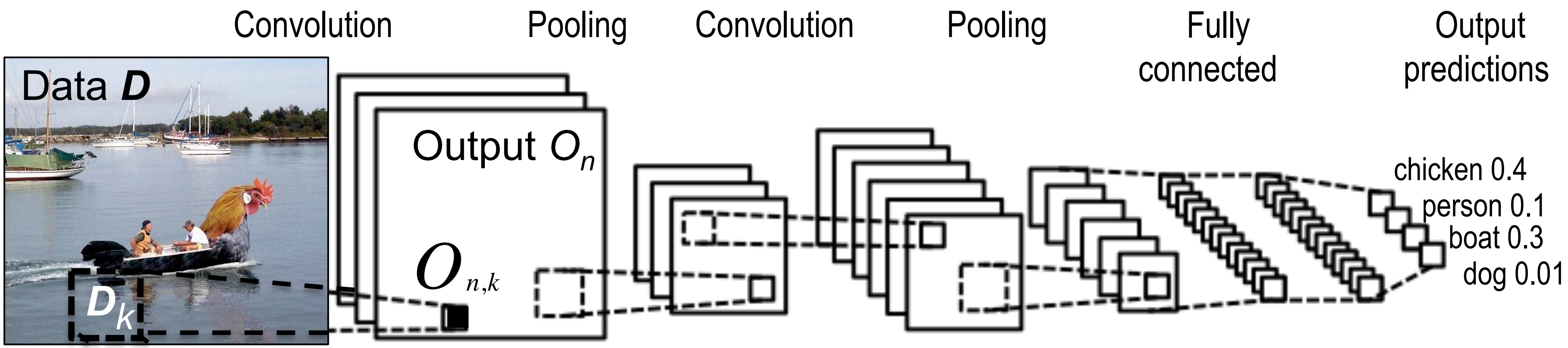
- **Direct solvers**
  - Factor and solve in working precision
- **Mixed Precision Iterative Refinement**
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g.
    $x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$

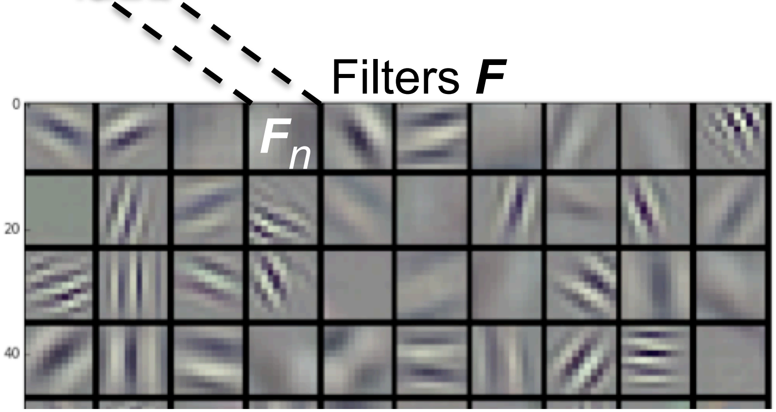
GPU TITAN X (3,072 CUDA cores @ 1.076 GHz)
Z/C GEMM peak ~ 190 / 5,600 GFlop/s; Maxwell
CPU Intel Xeon X5660@2.80GHz (2 x 6 cores)

Backend for DNN and Data Analytics

Support for various **Batched and/or Tensor contraction** routines

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:

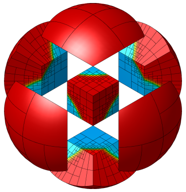
- For every filter F_n and every channel, the computation for every pixel value $O_{n,k}$ is a **tensor contraction**:

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

- 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)

Tensor contractions for high-order FEM

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

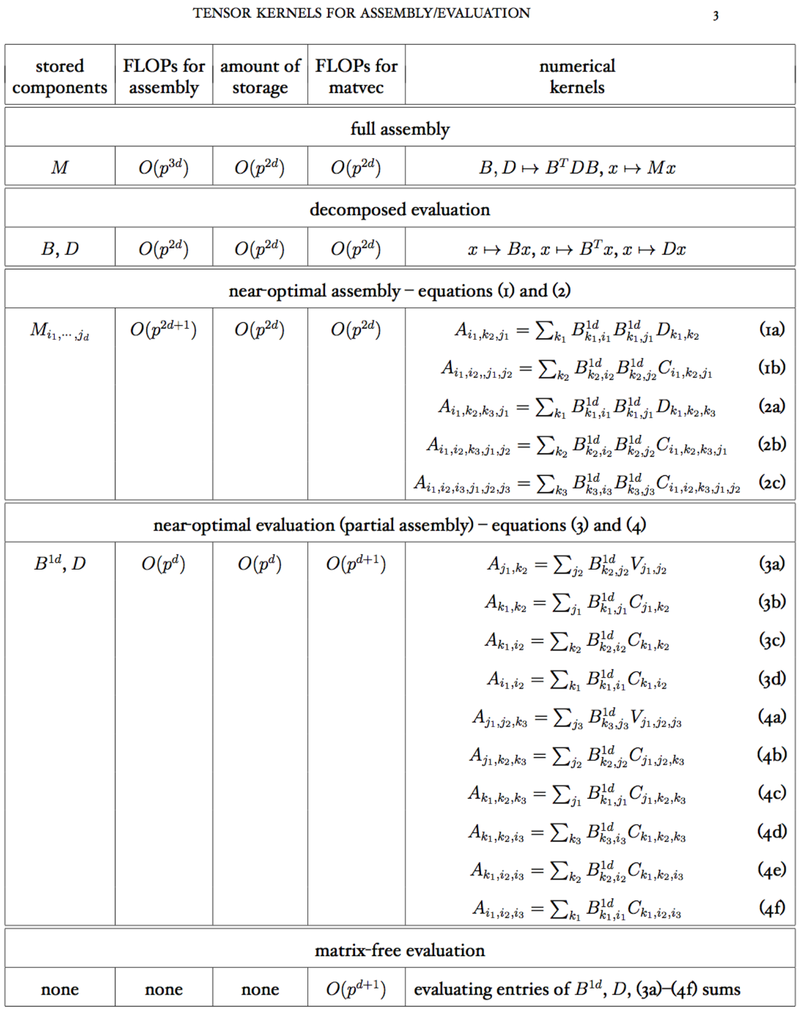
$$\textbf{Momentum Conservation:}\quad \frac{d\mathbf{v}}{dt} = -\mathbf{M}_{\mathbf{v}}^{-1}\mathbf{F}\cdot\mathbf{1}$$

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

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

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

Reference:
V. Dobrev, Tz. Kolev, R. Rieben, ***High-order curvilinear finite element methods for Lagrangian hydrodynamics***, SIAM J. Sci. Comp., B606-B641. (36 pages)



TENSOR KERNELS FOR ASSEMBLY/EVALUATION 3

stored components	FLOPs for assembly	amount of storage	FLOPs for matvec	numerical kernels
full assembly				
M	$O(p^{3d})$	$O(p^{2d})$	$O(p^{2d})$	$B, D \mapsto B^T D B,\ x \mapsto Mx$
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

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

Reference:
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,*** ICCS 2016, San Diego, CA, June 6—8, 2016.

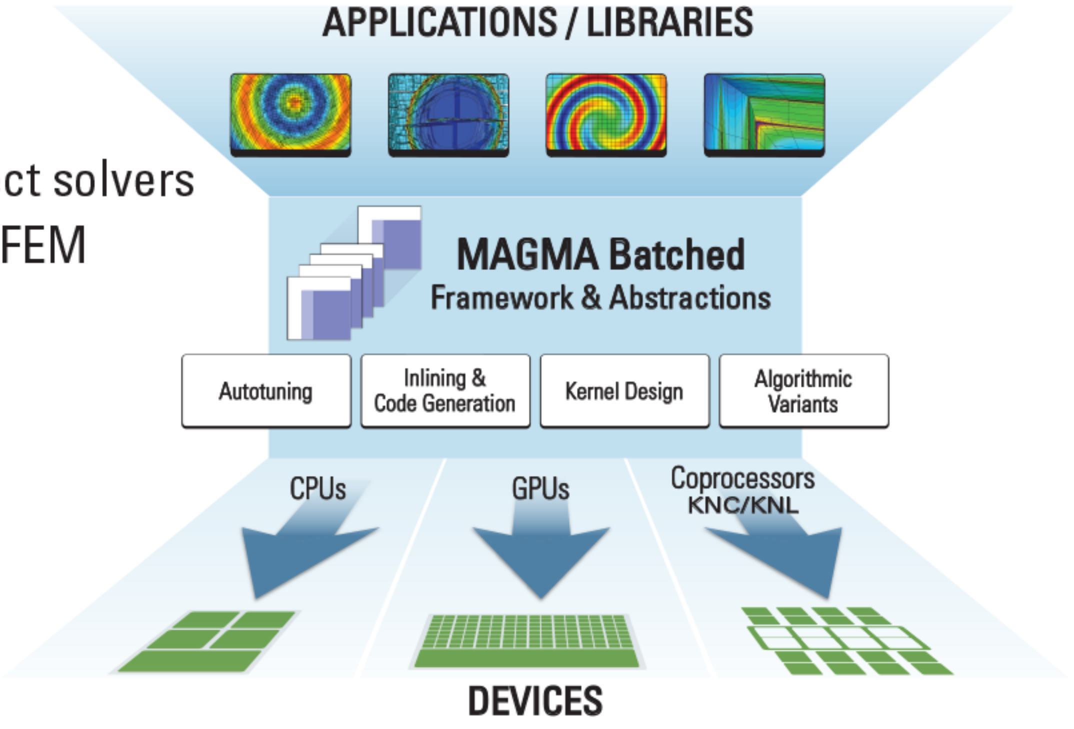
Batched routines released in MAGMA

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	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

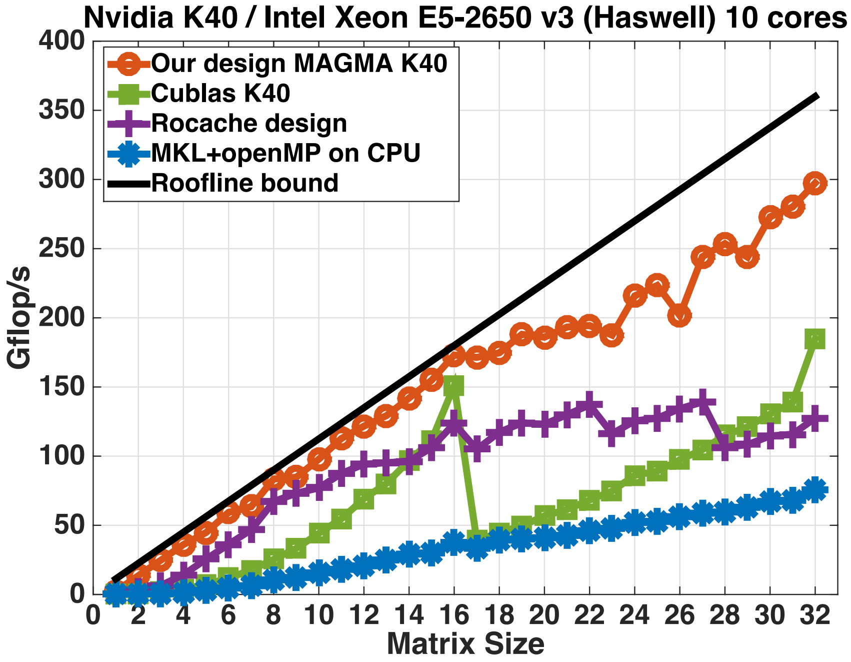
Batched BLAS Slides:
https://www.dropbox.com/s/kz4fhcipz3e56ju/BatchedBLAS-1.pptx?dl=0

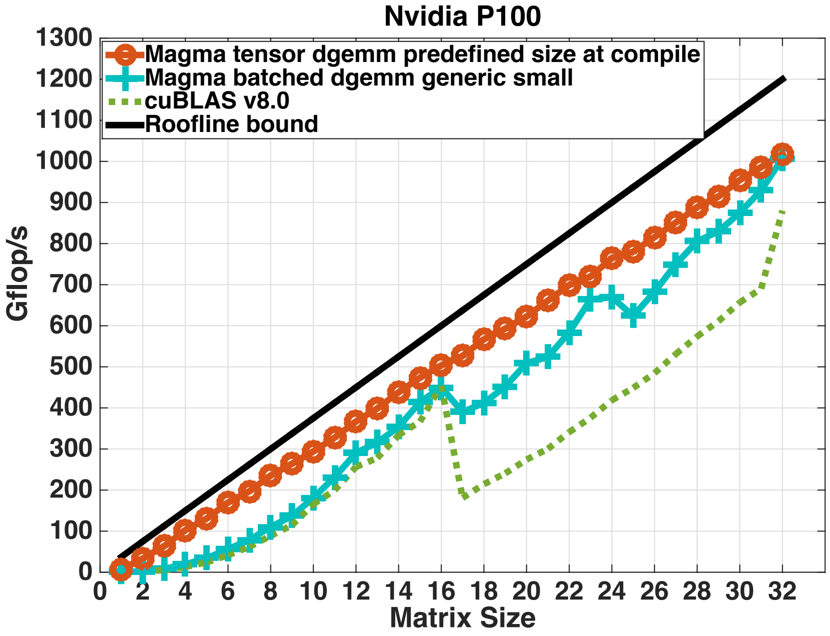
Webpage on ReproBLAS:
http://bebop.cs.berkeley.edu/reproblas/

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

Tensor contractions – performance

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.


Nvidia K40 / Intel Xeon E5-2650 v3 (Haswell) 10 cores


Nvidia P100

Reference:

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.

Motivation

- Deep learning architectures show promising results in abstract tasks like image classifications

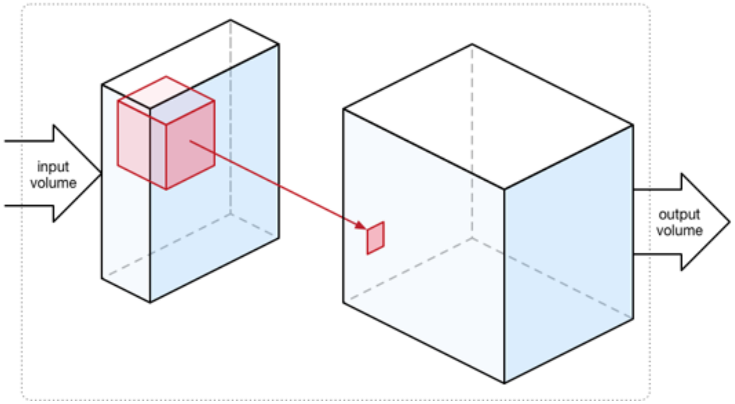
- They inherently consist of same old neural networks as before except
    - The size of the networks has increased drastically, more compute power,
    - Training dataset is huge, but fast

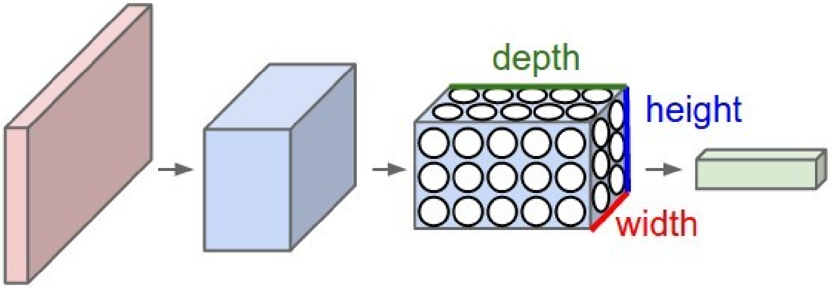
- Any improvement in the core modules of such networks will greatly influences the training and increases performance, also helps in understanding the network well

- Spatial convolution in convnets takes up to 70% of the total execution.

- We optimize spatial convolution module specifically for convnets.

Introduction to Spatial Convolution:





- Input and weights are 3D tensors
- As the filter traverses across the input volume horizontally and vertically it generates a 2D activation map
- Multiple filters generate multiple 2D output frames
- These output frames are stacked to form a 3D output tensor

Background or existing technique: Unfold and GEMM

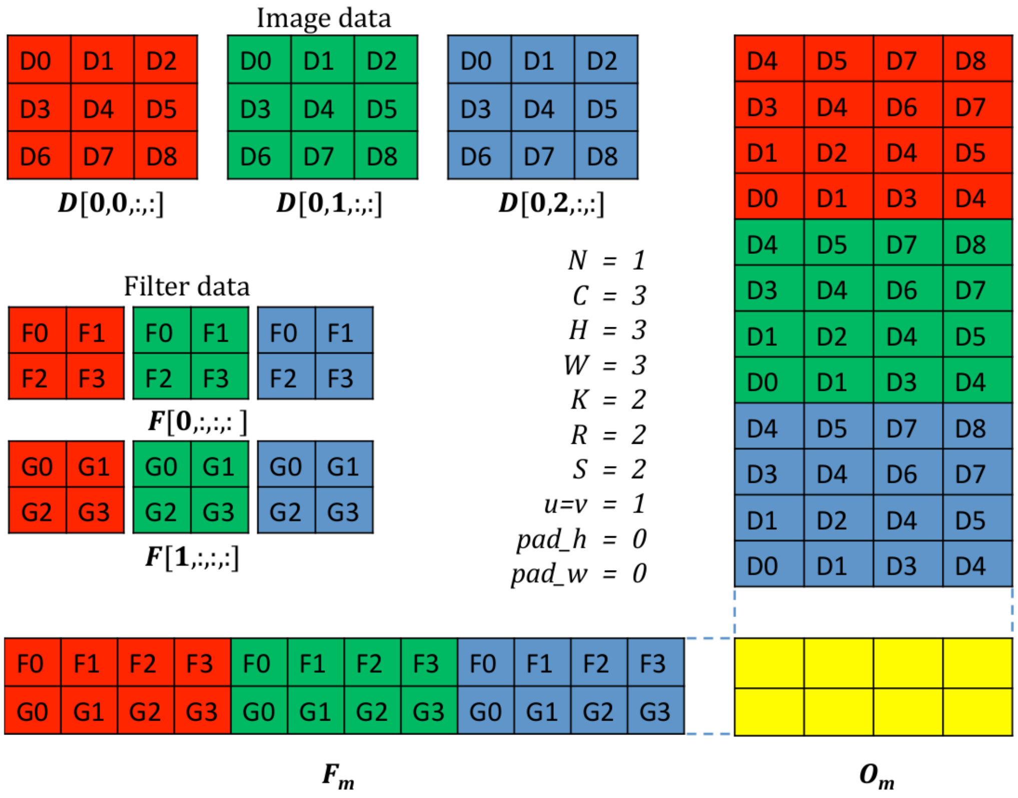


Image data

D0	D1	D2
D3	D4	D5
D6	D7	D8

D[0,0,:,:]

D0	D1	D2
D3	D4	D5
D6	D7	D8

D[0,1,:,:]

D0	D1	D2
D3	D4	D5
D6	D7	D8

D[0,2,:,:]

Filter data

F0	F1	F0	F1	F0	F1
F2	F3	F2	F3	F2	F3

F[0,:,:,:]

G0	G1	G0	G1	G0	G1
G2	G3	G2	G3	G2	G3

F[1,:,:,:]

$N = 1$
$C = 3$
$H = 3$
$W = 3$
$K = 2$
$R = 2$
$S = 2$
$u{=}v = 1$
$pad\_h = 0$
$pad\_w = 0$

F0	F1	F2	F3	F0	F1	F2	F3	F0	F1	F2	F3
G0	G1	G2	G3	G0	G1	G2	G3	G0	G1	G2	G3

F_m

D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4

O_m

<u>VGG-16 D conv modules</u>

Conv	Weight Matrix	Data Matrix
1	64x27	27x50176
2	64x576	576x50176
3	128x576	576x12544
4	128x1152	1152x12544
5-7	256x2304	2304x3136
8	512x2304	2304x784
9-10	512x4608	4608x784
10-13	512x4608	4608x196

Background or existing technique: Unfold and GEMM

Advantages:

- Unfold involves streaming memcpy and can be made parallel by having many threads working on many sections of the input
- Many BLAS libraries contain fine tuned GEMM routines that can be used
- The output format is consistent with the actual convolution output

Disadvantages:

- Unfold operation requires extra memory
- Matrix shapes can be greatly skewed

Convolution using transformation techniques

FFT method:

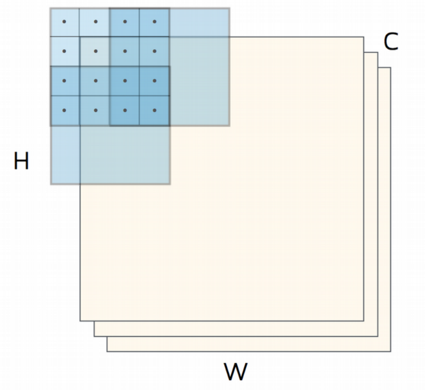
- Convolution becomes elementwise product in frequency domain
- Complexity in 2D is O(RS log(HW)) , better than O(RSHW) in direct convolution
- Caution !! filter dimension should be similar to image dimension but in convnets that are used widely RS << HW

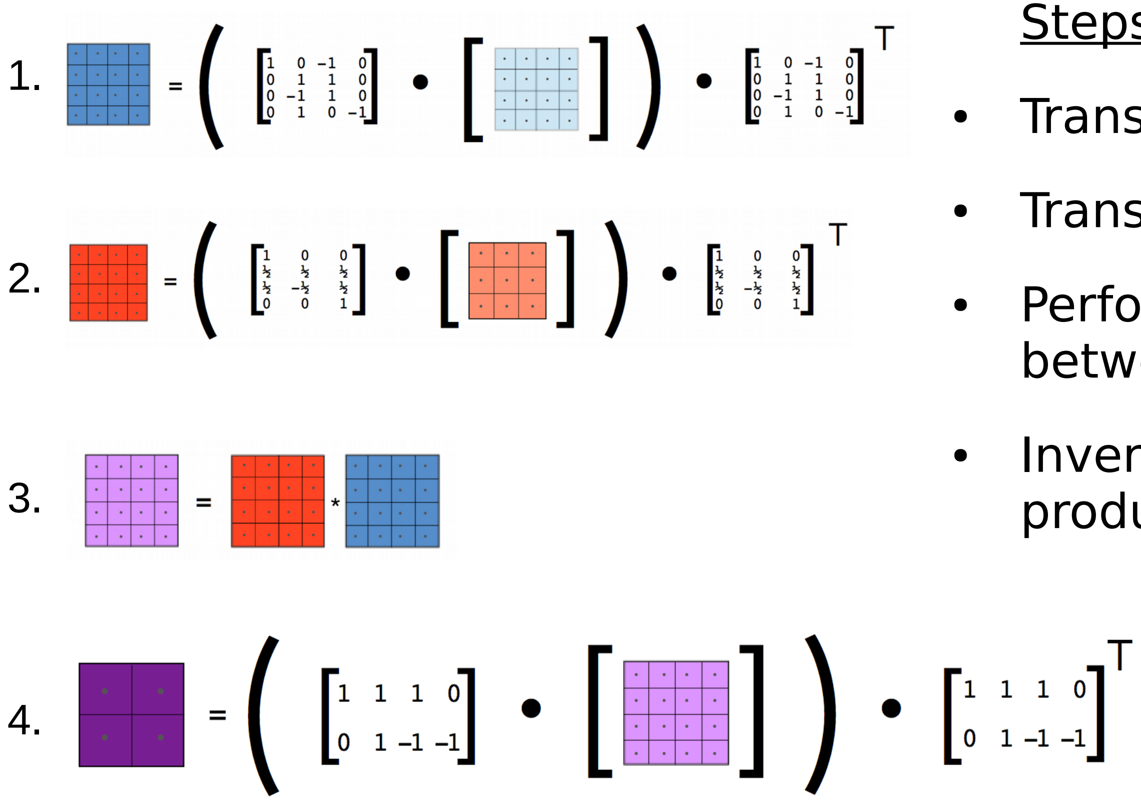
Winograd Minimal filtering:

- Best suited for small filters, RS << HW
- Reduces the arithmetic operations by constant factor thus improves the asymptotic timing

MAGMA-DNN

Winograd algorithm



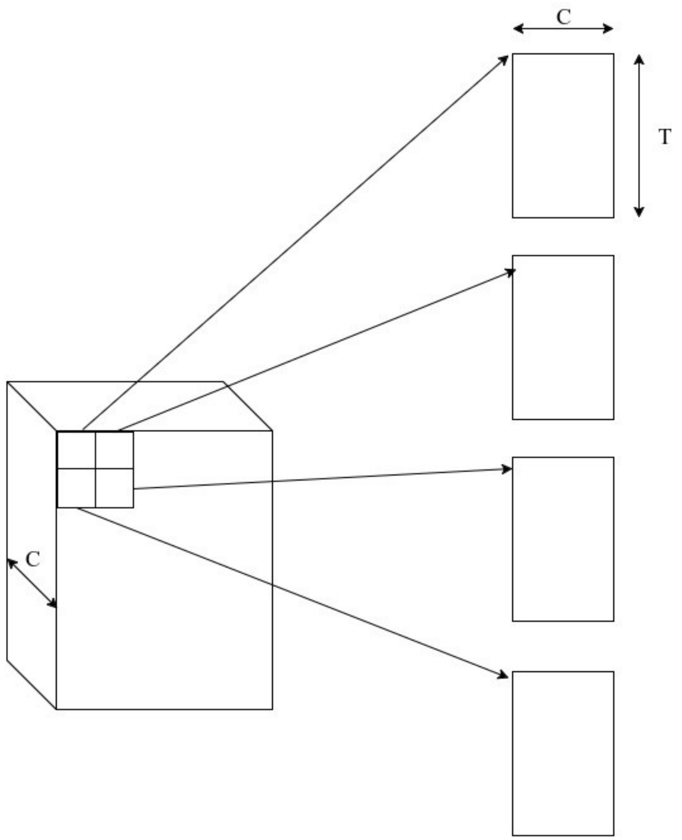


Steps:

- Transform a 4x4 image tile
- Transform a 3x3 filter
- Perform element wise product between the transformed tiles
- Inverse transform on the product tile

MAGMA-DNN

Winograd algorithm: reduction to GEMM’s



- Each Image tensor has 16 matrices of size TxC
- The K filters are reduced to 16 CxK matrices
- For a batch size of N there are 16N GEMMs, for example N=64 gives 1024 GEMMs.
- 16 filter matrices are common for all the GEMMs, so better last level cache efficiency
- Once GEMM are performed, “Gather” the elements from the 16 output matrices to form a 4x4 output tile
- Apply inverse transform on the output tile to obtain 2x2 convolution output

Winograd algorithm: Advantage of GEMM's

- Each transformed image tile is reused with K filters. Similarly, each filter tile is reused with all the input tiles across the batch of N

- If N and K are large enough, the transformation cost is amortized because of max re- usage of transformed tiles

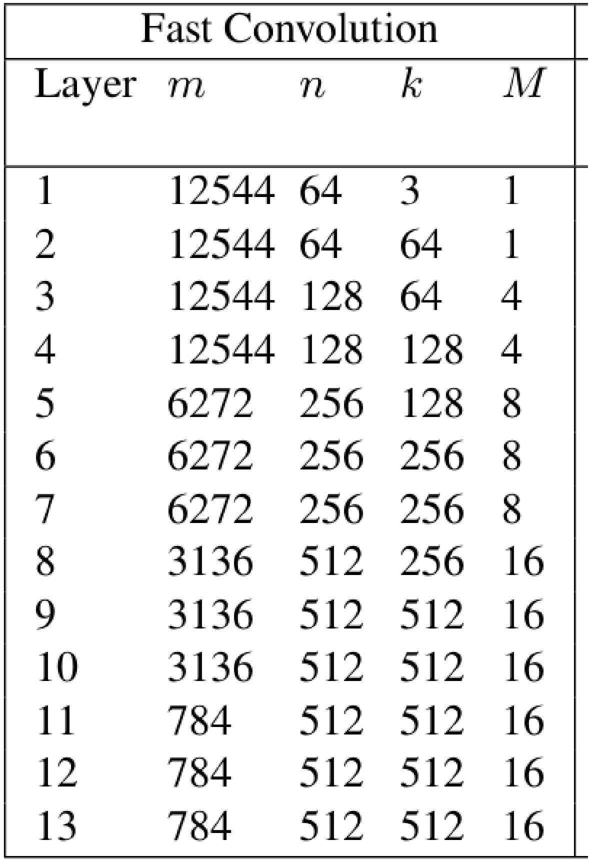
- Instead of applying inverse across C and then accumulating, the natural form of GEMM accumulates the result across C and then inverse can be applied once to the GEMM output tile.

- This is possible because of the linearity property for Winograd convolution Fine tuned GEMM APIs are available

- Good cache efficiency Good arithmetic intensity

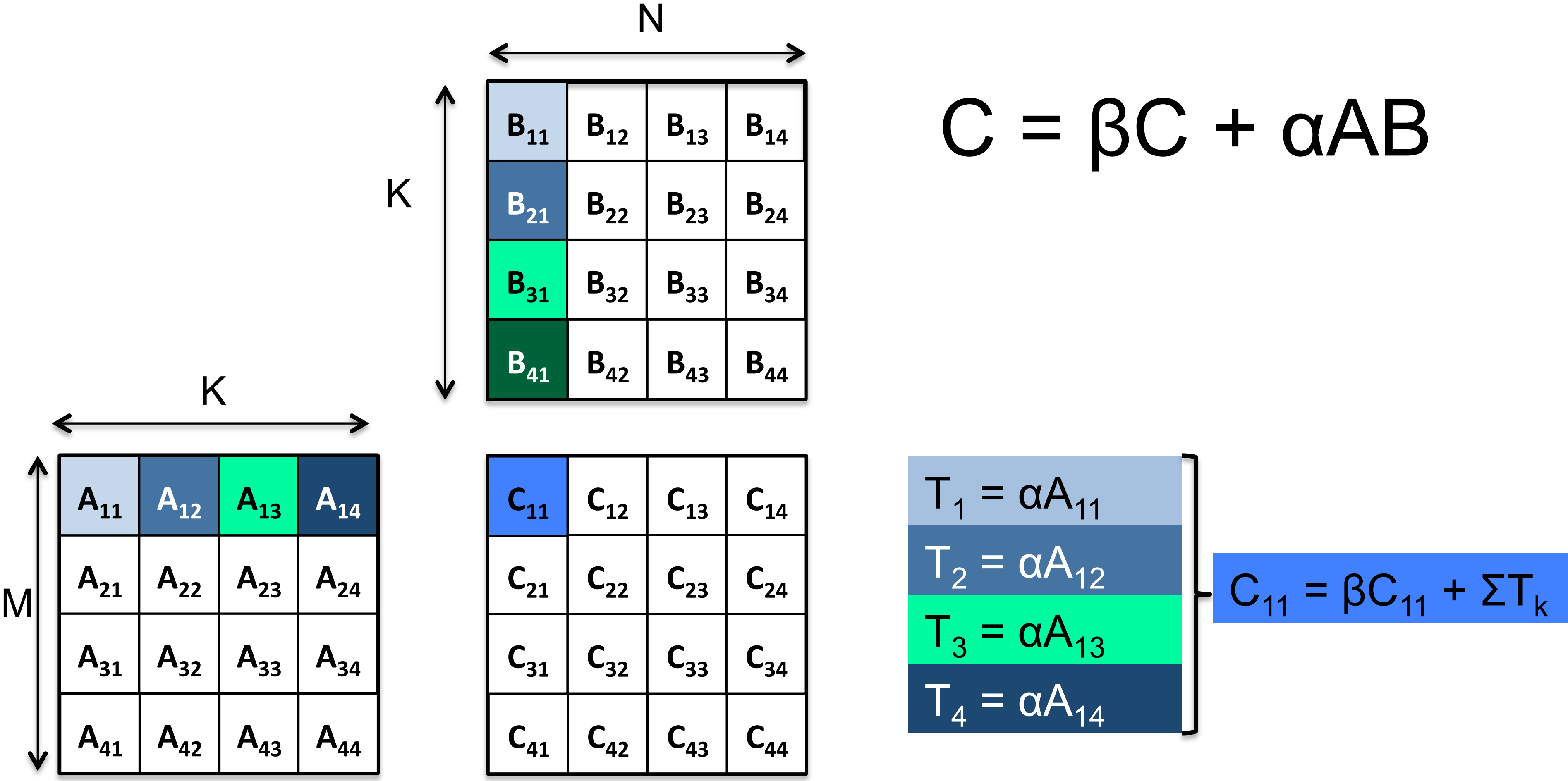
MAGMA-DNN

Winograd algorithm



Fast Convolution				
Layer	m	n	k	M
1	12544	64	3	1
2	12544	64	64	1
3	12544	128	64	4
4	12544	128	128	4
5	6272	256	128	8
6	6272	256	256	8
7	6272	256	256	8
8	3136	512	256	16
9	3136	512	512	16
10	3136	512	512	16
11	784	512	512	16
12	784	512	512	16
13	784	512	512	16

Optimizing GEMM's: Kernel design



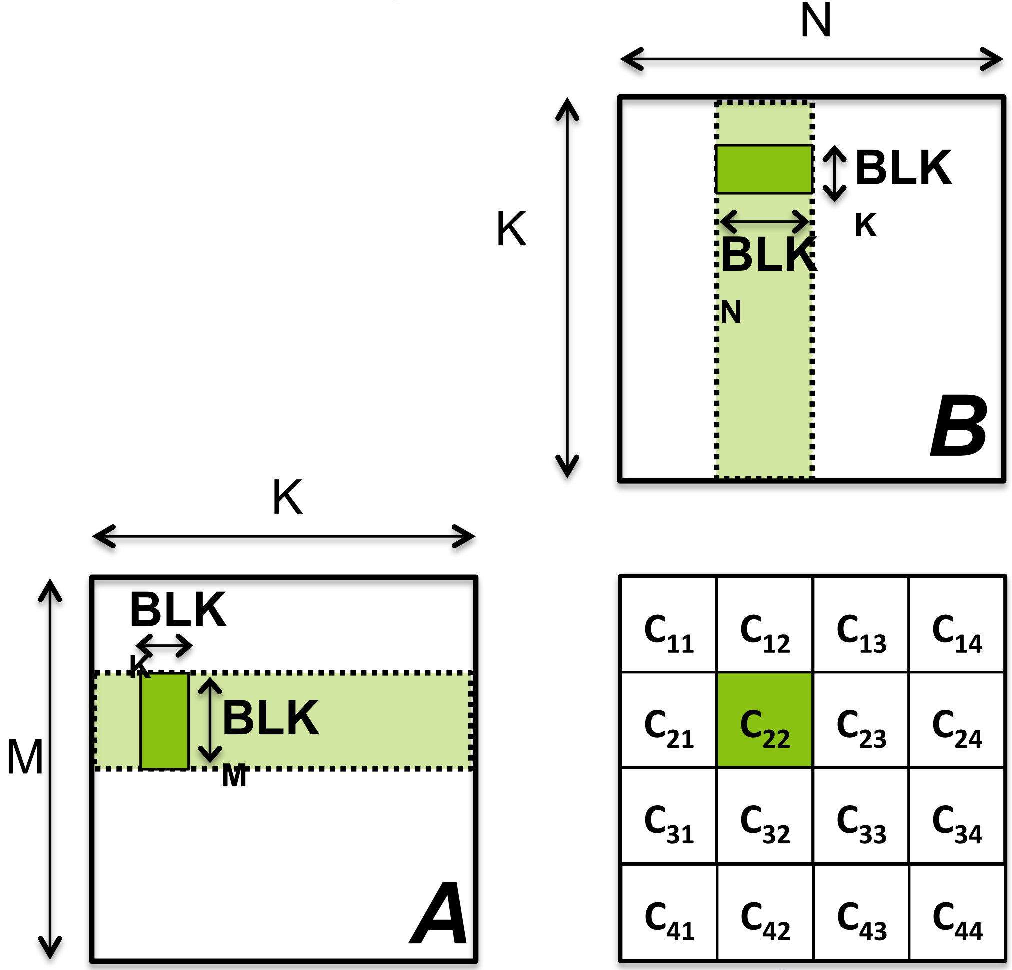
$$C = \beta C + \alpha AB$$

$$\begin{array}{c|cccc} & & N & & \\ \hline K & B_{11} & B_{12} & B_{13} & B_{14} \\ & B_{21} & B_{22} & B_{23} & B_{24} \\ & B_{31} & B_{32} & B_{33} & B_{34} \\ & B_{41} & B_{42} & B_{43} & B_{44} \end{array}$$

$$\begin{array}{c|cccc} & & K & & \\ \hline M & A_{11} & A_{12} & A_{13} & A_{14} \\ & A_{21} & A_{22} & A_{23} & A_{24} \\ & A_{31} & A_{32} & A_{33} & A_{34} \\ & A_{41} & A_{42} & A_{43} & A_{44} \end{array} \qquad \begin{array}{cccc} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{array}$$

$$\left.\begin{array}{l} T_1 = \alpha A_{11} \\ T_2 = \alpha A_{12} \\ T_3 = \alpha A_{13} \\ T_4 = \alpha A_{14} \end{array}\right\} C_{11} = \beta C_{11} + \Sigma T_k$$

Optimizing GEMM's: Kernel design

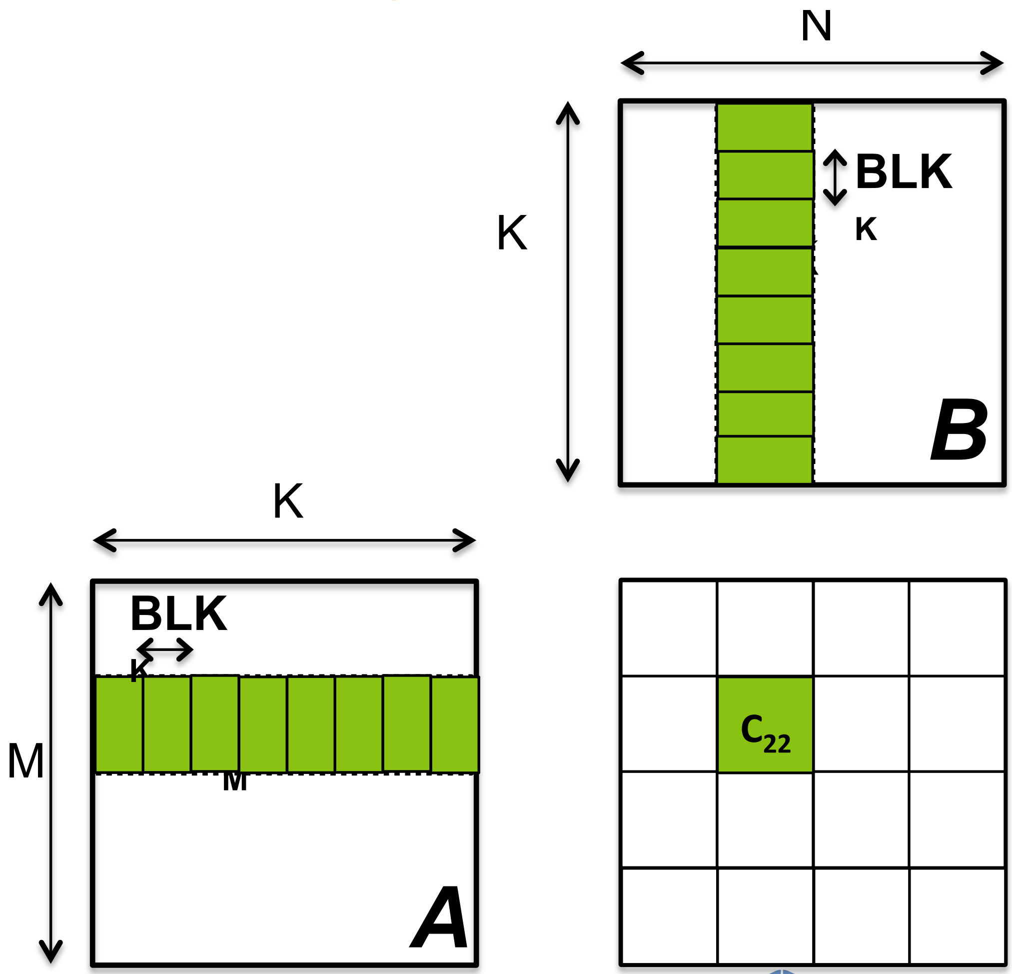


C_{11}	C_{12}	C_{13}	C_{14}
C_{21}	C_{22}	C_{23}	C_{24}
C_{31}	C_{32}	C_{33}	C_{34}
C_{41}	C_{42}	C_{43}	C_{44}

$$C = \beta C + \alpha AB$$

- Assign every block of C_{ij} to a TB
- Hold the block C_{ij} in register/sm
- Slide the **green tile** of A and B and compute $C = \beta C + \alpha AB$
- **This design guarantee reproducibility of results**
- **The kernel is parameterized to allow tuning and optimization**

Optimizing GEMM's: Kernel design



$$C = \beta C + \alpha AB$$

$$T \;\; = A_1 \, B_1$$

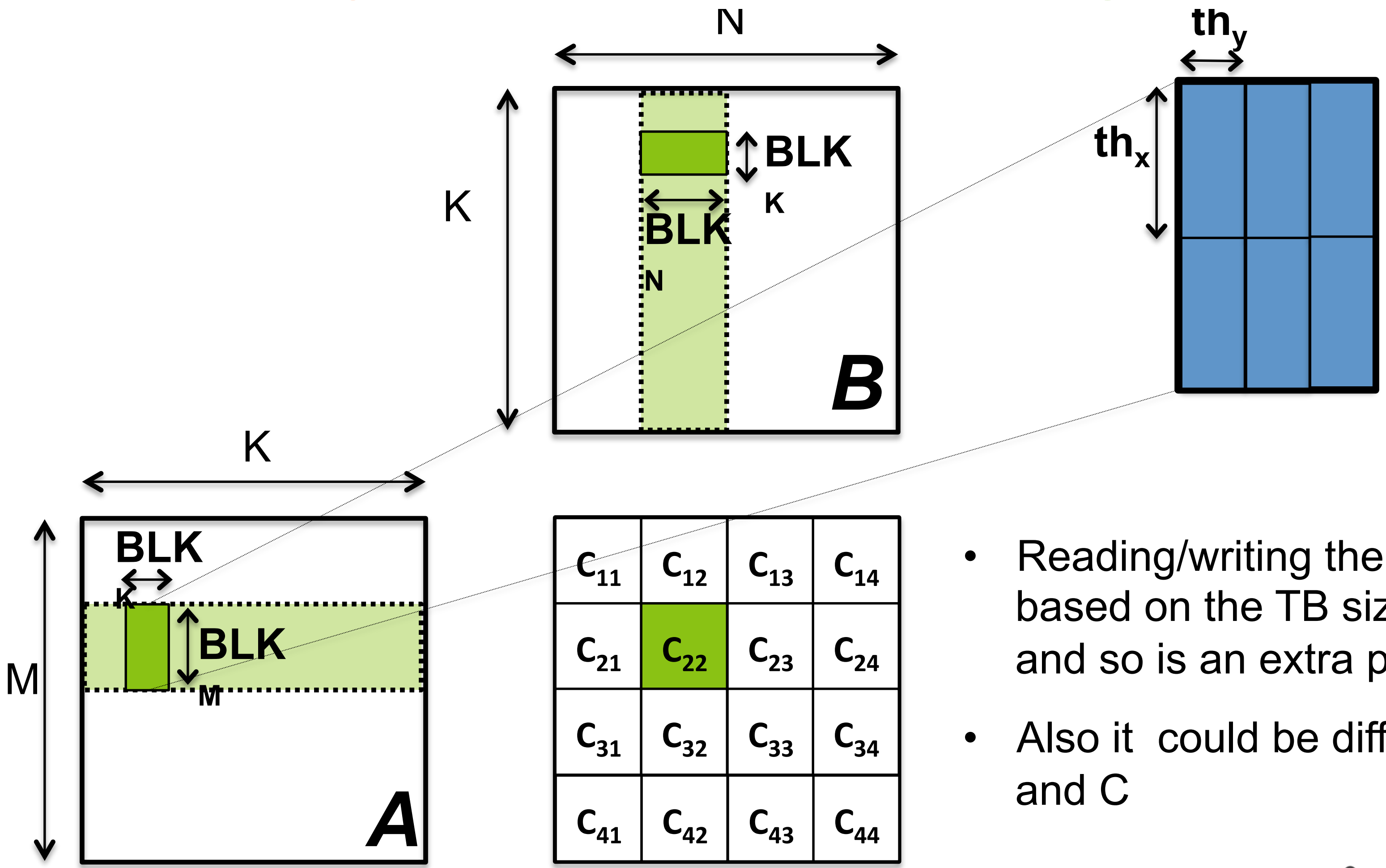
$$T \mathrel{+}= A_2 \, B_2$$

$$T \mathrel{+}= A_3 \, B_3$$

$$T \mathrel{+}= A_4 \, B_4$$

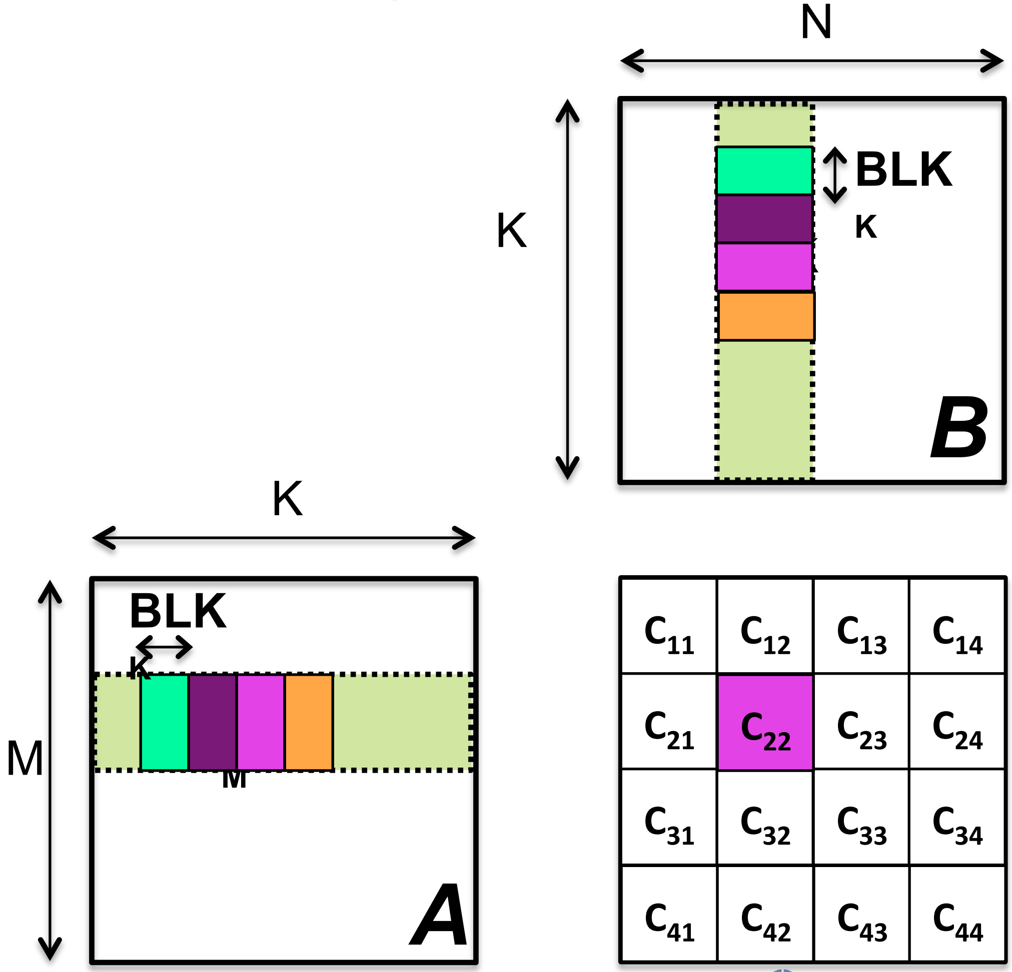
$$C_{22} = \beta C_{22} + \alpha \, T$$

Optimizing GEMM's: Kernel design



- Reading/writing the elements is based on the TB size (# threads) and so is an extra parameter.
- Also it could be different for A, B and C

Optimizing GEMM's: Kernel design



- Prefetching

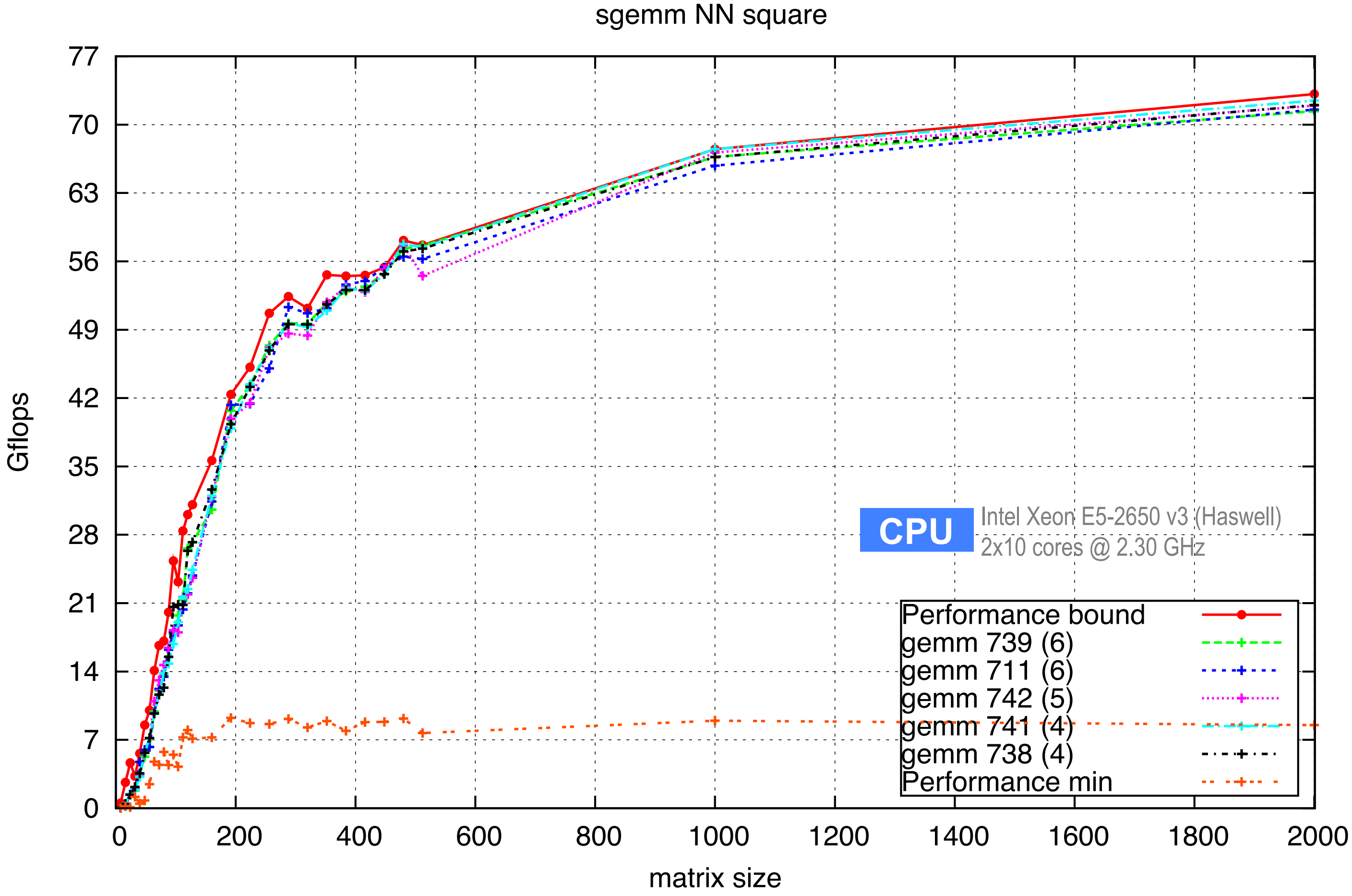
Optimizing GEMM's: Kernel design

Are we done, we have our best kernel ?

- for most of the case LA algorithms, Deep Learning, etc., the matrices A,B,C are not squares which requires autotunning

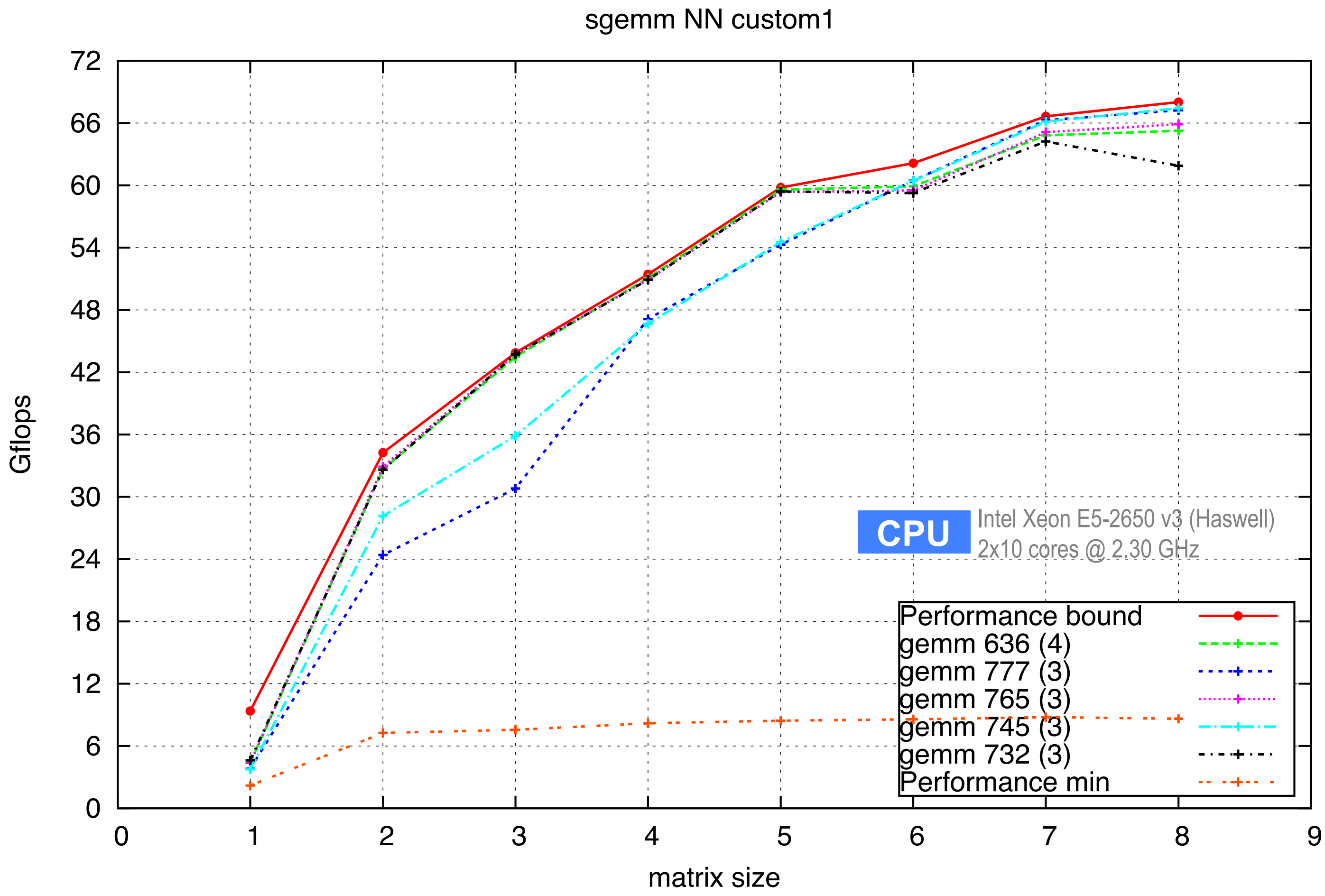
MAGMA-DNN

Optimizing GEMM's: Kernel tuning



CPU Intel Xeon E5-2650 v3 (Haswell)
2x10 cores @ 2.30 GHz

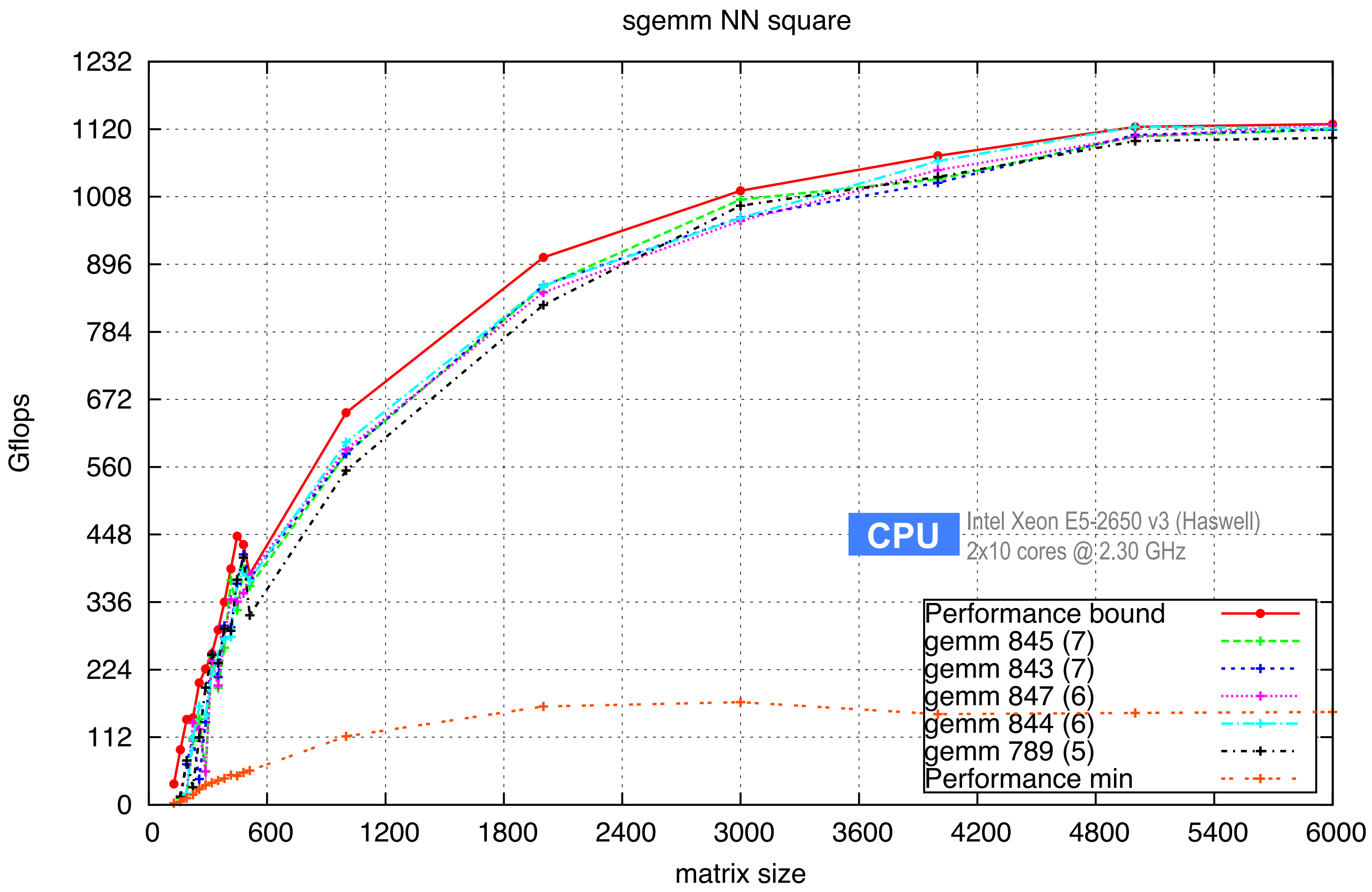
Optimizing GEMM's: Kernel tuning



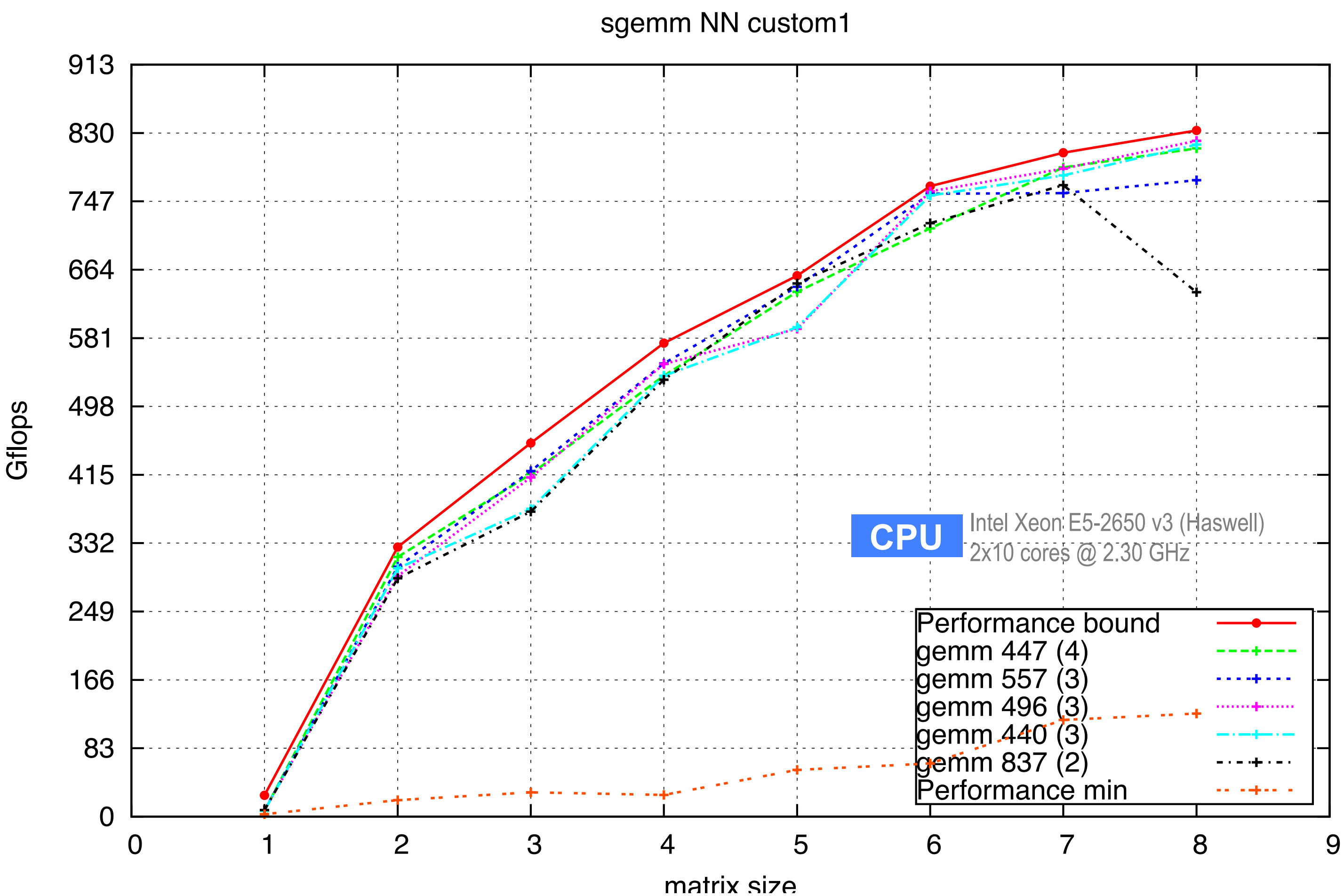
CPU Intel Xeon E5-2650 v3 (Haswell)
2x10 cores @ 2.30 GHz

MAGMA-DNN

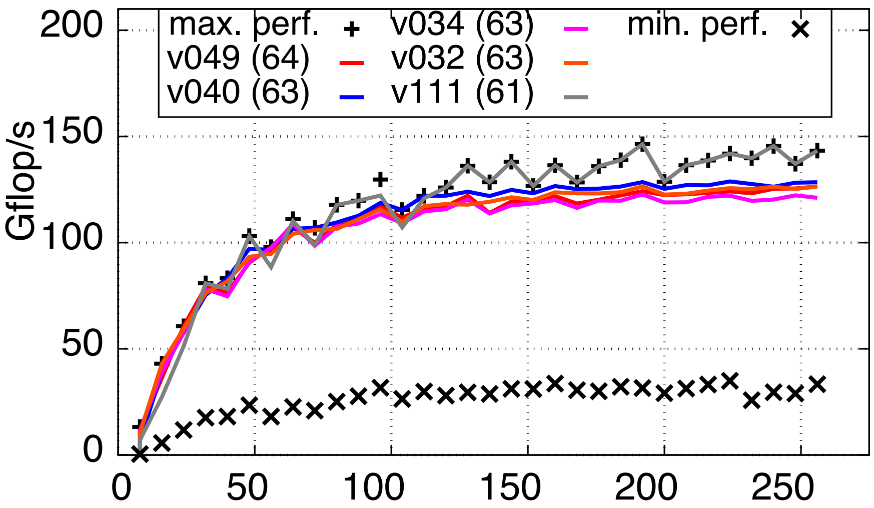
Optimizing GEMM's: Kernel tuning



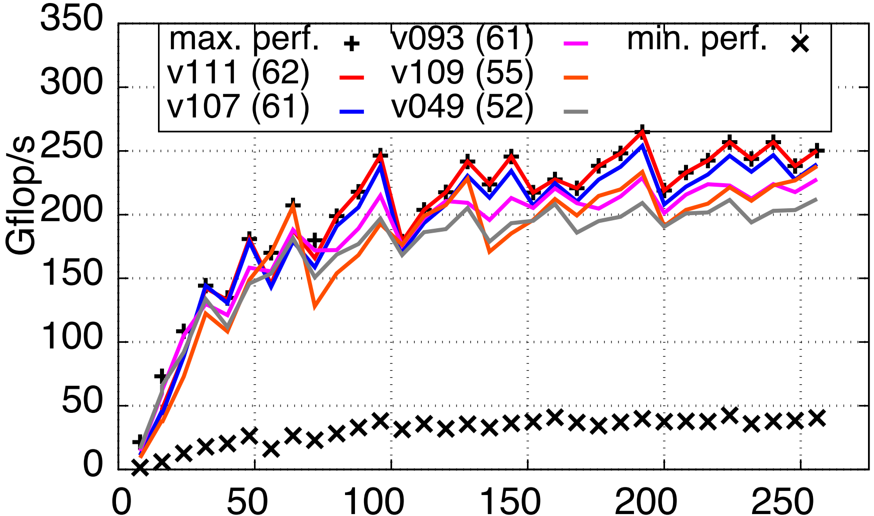
Optimizing GEMM's: Kernel tuning



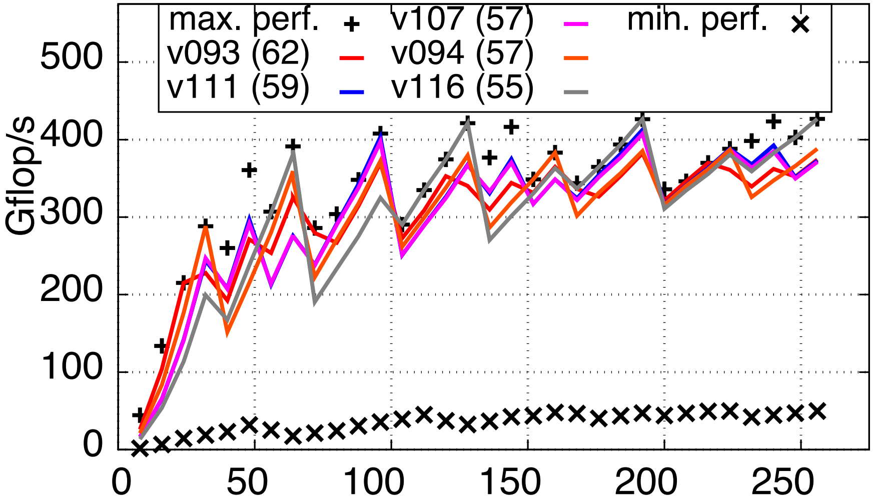
CPU Intel Xeon E5-2650 v3 (Haswell)
2x10 cores @ 2.30 GHz



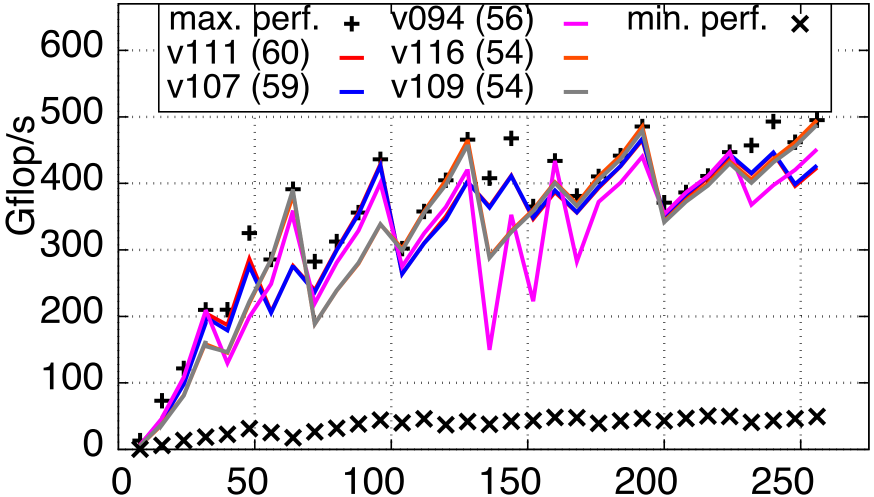
m=n, k=8



m=n, k=16



m=n, k=64



m=n=k

K40 NVIDIA K40 GPU
15 MP x 192 @ 0.88 GHz

Conclusions and future work

In conclusion:

- Developed a number of NLA in MAGMA targeting applications
  - High-order FEM, DNN, and data analytics;
  - Tensor abstractions and high-performance tensor contractions (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
- To release NLA backend for DLA and data analytics
- Integrate developments in applications
- Complete autotuning and develop all kernels

Collaborators and Support

MAGMA team

http://icl.cs.utk.edu/magma

Collaborating partners

University of Tennessee, Knoxville
Lawrence Livermore National Laboratory,
    Livermore, CA
    LLNL led ECP CEED:
    Center for Efficient Exascale Discretizations
University of Manchester, Manchester, UK
University of Paris-Sud, France
INRIA, France

Umeå University

INRIA

Rutherford Appleton Laboratory

University of Manchester