MagmaDNN 0.2 High-Performance Data Analytics for Manycore GPUs and CPUs

Presentation · January 2019

DOI: 10.13140/RG.2.2.14906.64961

11 authors, including:

Lucien Ng
The Chinese University of Hong Kong
4 PUBLICATIONS 5 CITATIONS

Alex Gessinger
Slippery Rock University of Pennsylvania
3 PUBLICATIONS 0 CITATIONS

MagmaDNN 0.2

High-Performance Data Analytics for Manycore GPUs and CPUs

Lucien Ng[1], Sihan Chen[1], Alex Gessinger[4], Daniel Nichols[3], Sophia Cheng[1], Anu Meenasorna[2]
[1] The Chinese University of Hong Kong
[2] National Institute of Technology
[3] The University of Tennessee, Knoxville (UTK)
[4] Slippery Rock University

Kwai Wong[1,2], Stanimire Tomov[3], Azzam Haidar[4], Ed D'Azevedo[2], Jack Dongarra[3,2]
[1] The Joint Institute for Computational Sciences (JICS), UTK
[2] Oak Ridge National Laboratory (ORNL)
[3] The Innovative Computing Laboratory, UTK
[4] Nvidia Corporation

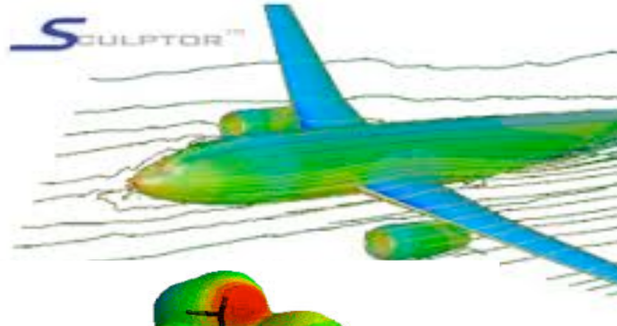
Summer Research Experiences for Undergraduate (REU)
Research Experiences in Computational Science, Engineering, and Mathematics (RECSEM)
Knoxville, TN

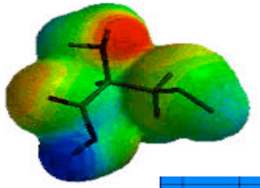
The University of Tennessee
Department of Electrical Engineering and Computer Science

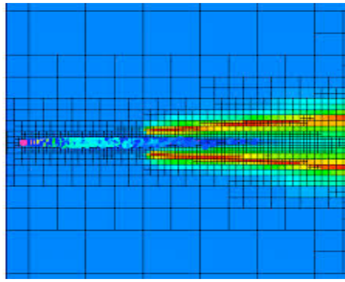
Dense Linear Algebra in Applications

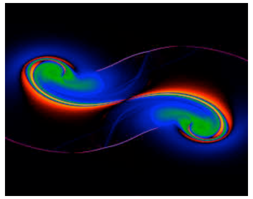
Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:

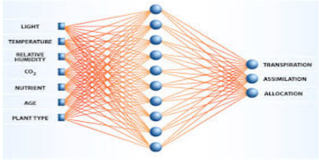
- **Linear systems:** Solve $Ax = b$
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** Find x to minimize $\| Ax - b \|$
  - Computational 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
- **SVD:** $A = U \Sigma V^*$ ($Au = \sigma v$ and $A^*v = \sigma u$)
  - 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.
- **DLA is crucial to the development of sparse solvers**











Dense Linear Algebra in Applications

Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:

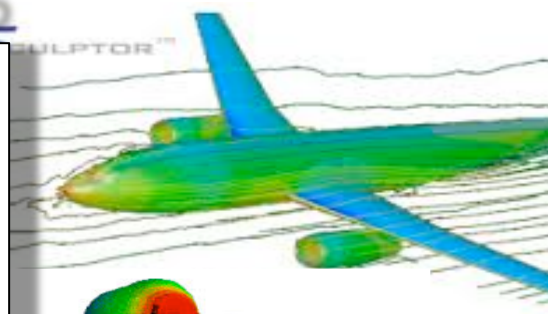
- **Linear systems:** Solve $Ax = b$
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** Find x to minimize $\| Ax - b \|$
  - Computational 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
- **SVD:** $A = U \Sigma V^*$ ($Au = \sigma v$ and $A^* v = \sigma u$)
  - 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.
- **DLA is crucial to the development of sparse solvers**

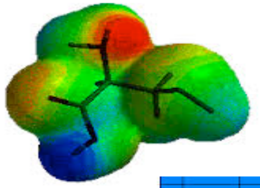
Provided in MAGMA 2.5

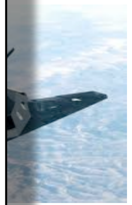
FEATURES AND SUPPORT

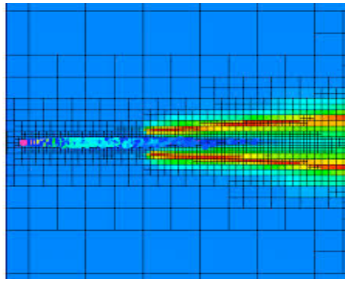
- MAGMA 2.5 FOR CUDA
- clMAGMA 1.4 FOR OpenCL
- MAGMA MIC 1.4 FOR Intel Xeon Phi

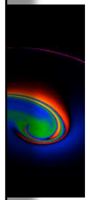
CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●	●		Auxiliary BLAS
●			Batched LA
●		●	Sparse LA
●	●	●	CPU/GPU Interface
●	●	●	Multiple precision support
● NEW			Mixed precision (including FP16)
●			Non-GPU-resident factorizations
● NEW			GPU-only factorizations
●	●	●	Multicore and multi-GPU support
● NEW			MAGMA Analytics/MagmaDNN 0.2
●	●	●	LAPACK testing
●	●	●	Linux
●	●		Windows
●	●		Mac OS

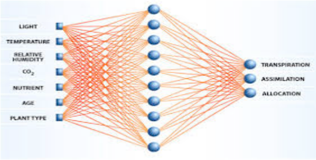












MAGMA

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

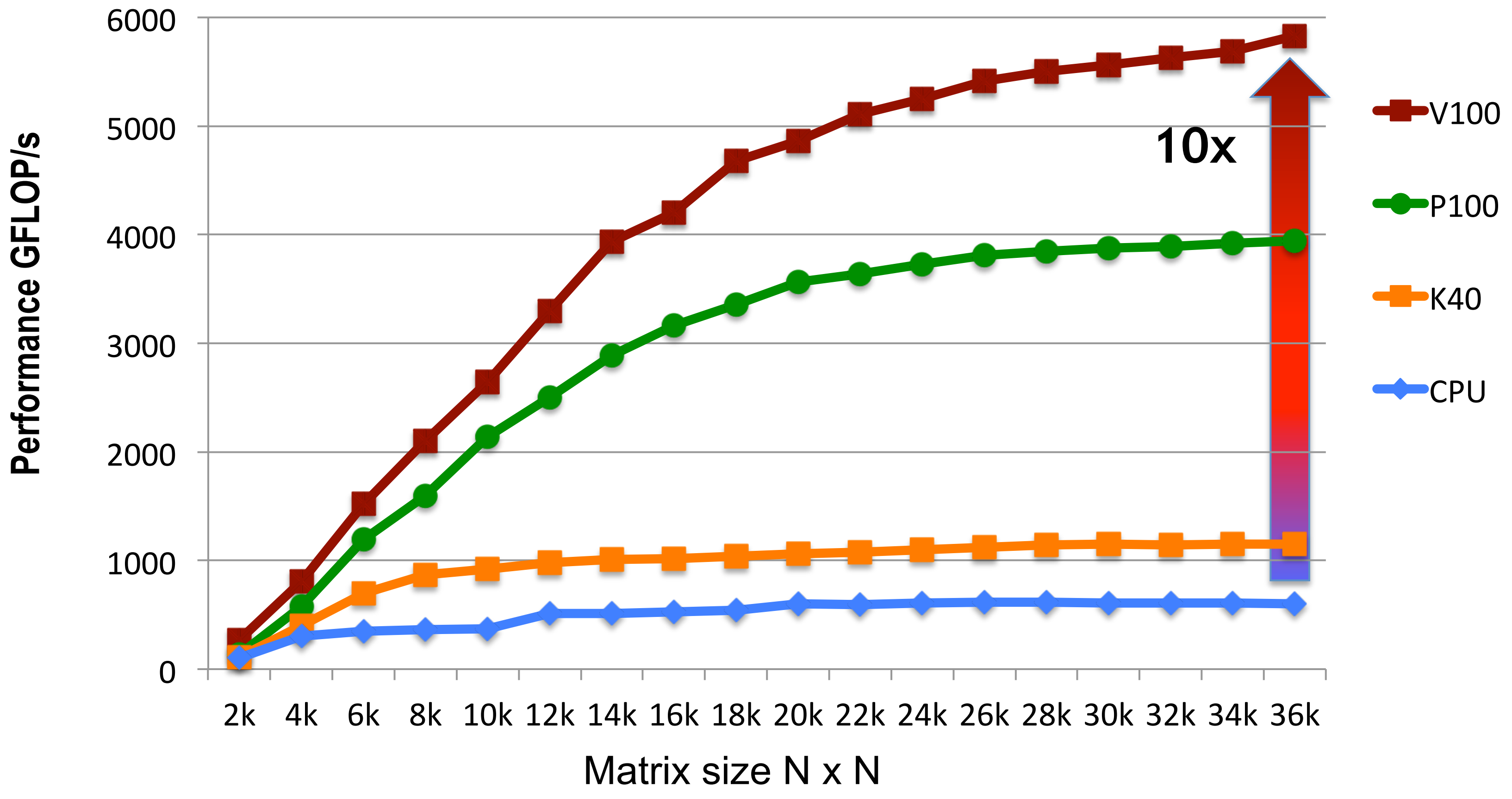
https://bitbucket.org/icl/magma

Why use GPUs in HPC?

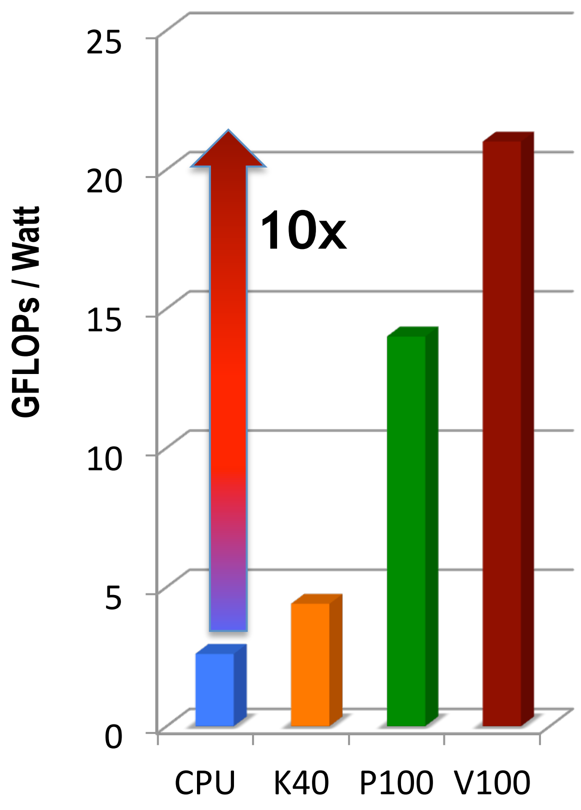
PERFORMANCE & ENERGY EFFICIENCY

MAGMA 2.5 LU factorization in double precision arithmetic

CPU Intel Xeon E5-2650 v3 (Haswell) 2x10 cores @ 2.30 GHz
K40 NVIDIA Kepler GPU 15 MP x 192 @ 0.88 GHz
P100 NVIDIA Pascal GPU 56 MP x 64 @ 1.19 GHz
V100 NVIDIA Volta GPU 80 MP x 64 @ 1.38 GHz



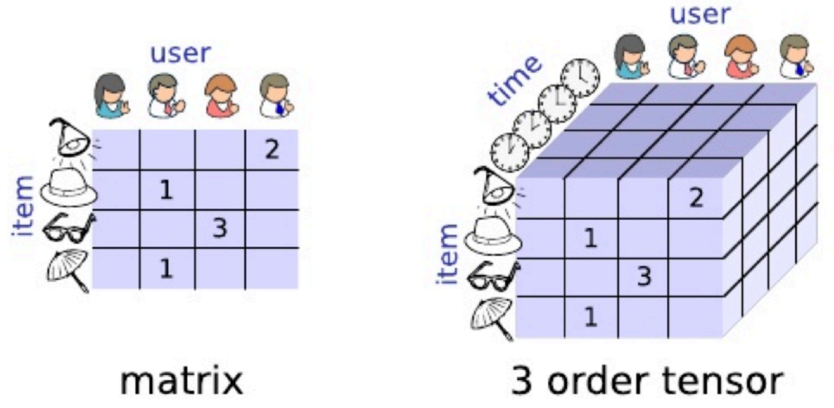
Energy efficiency
(under ~ the same power draw)



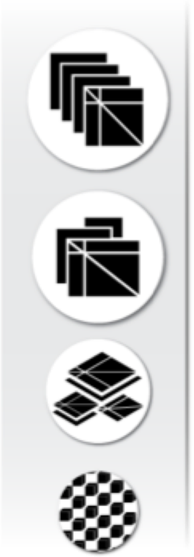
What about accelerated LA for Data Analytics?

- Traditional libraries like MAGMA can be used as backend to accelerate the LA computations in data analytics applications
- Need support for
  1) New data layouts, 2) Acceleration for small matrix computations, 3) Data analytics tools

Need data processing and analysis support for
Data that is multidimensional / relational



Small matrices, tensors, and batched computations



Fixed-size batches

Variable-size batches

Dynamic batches

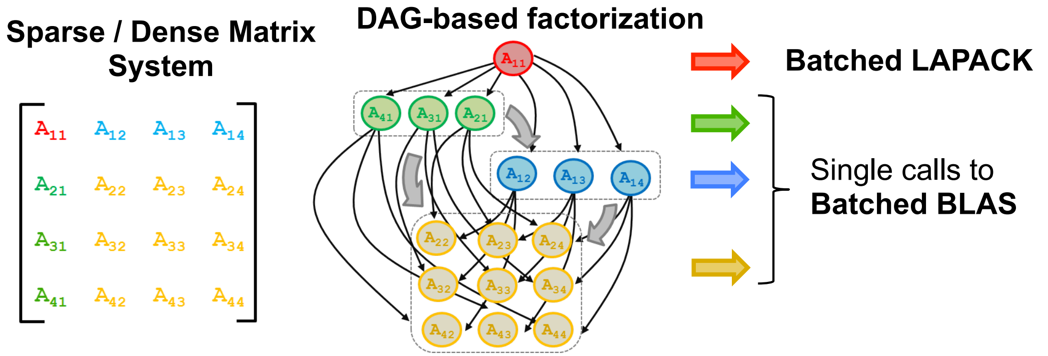
Tensors

Data Analytics and LA on many small matrices

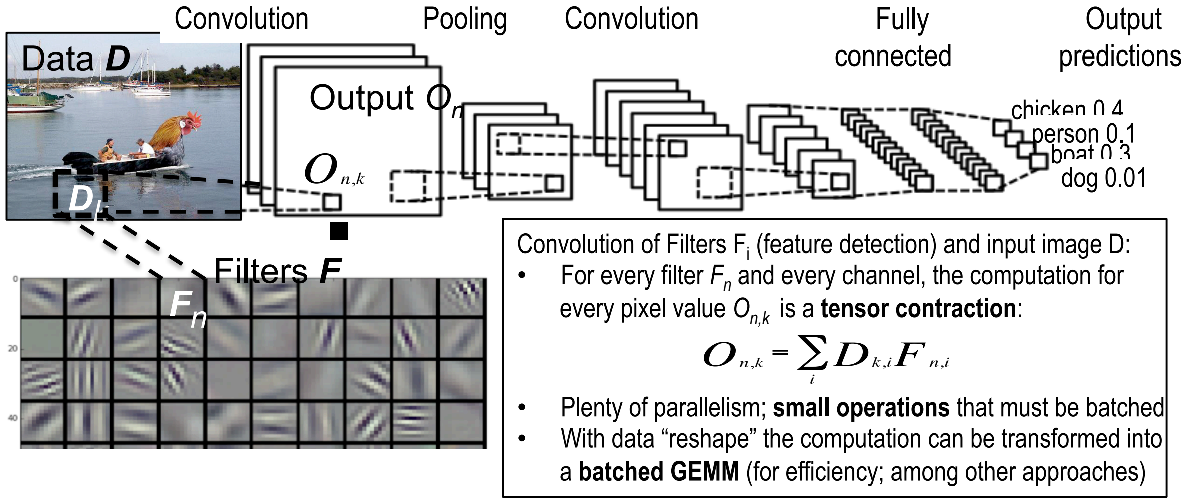
Data Analytics and associated with it Linear Algebra on small LA problems are needed in many applications:

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

Sparse/Dense solvers & preconditioners



Machine learning



Applications using high-order FEM

- Matrix-free basis evaluation needs efficient tensor contractions,

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

- **Within ECP CEED Project,** designed MAGMA batched methods to split the computation in many small high-intensity GEMMs, grouped together (batched) for efficient execution:

Batch_{ $C_{i3} = A^T B_{i3}$, for range of i3 }

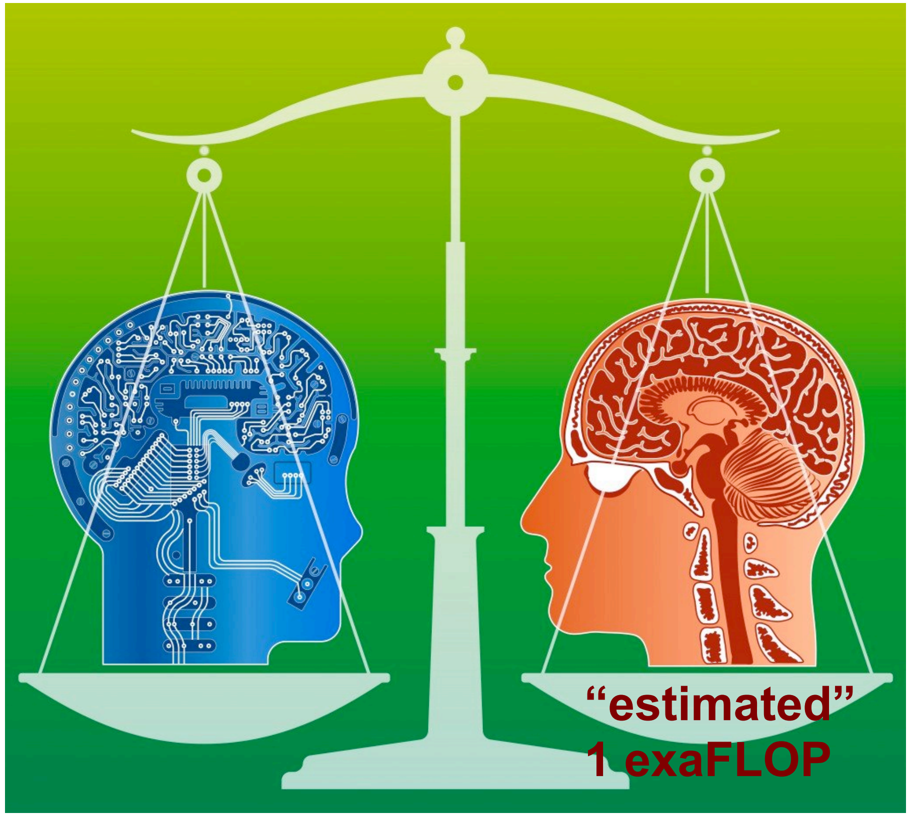
Machine learning / Artificial Intelligence

- Give computers the ability to “learn”

- Soon we may not have to program computers
  – We will train them instead !

See part of GTC’18 Keynote from NVIDIA CEO Jensen Huang
https://www.youtube.com/watch?v=oa__wkSmWUw

Human brain vs. supercomputer ?



https://www.scienceabc.com/humans/the-human-brain-vs-supercomputers-which-one-wins.html

MagmaDNN – Data Analytics Tool

- **MagmaDNN 0.2 – HP Data analytics and ML**
  GPU-accelerated numerical software using MAGMA as computational backend to accelerate its LA computations

- **Open source; looking for feedback and contributions**
  Started with students from REU/RECSEM program
  https://bitbucket.org/icl/magmadnn

Provided in MAGMA 2.5

FEATURES AND SUPPORT

- **MAGMA 2.5** FOR **CUDA**
- **clMAGMA 1.4** FOR **OpenCL**
- **MAGMA MIC 1.4** FOR **Intel Xeon Phi**

CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●	●		Auxiliary BLAS
●			Batched LA
●		●	Sparse LA
●	●	●	CPU/GPU Interface
●	●	●	Multiple precision support
● NEW			Mixed precision (including FP16)
●			Non-GPU-resident factorizations
● NEW			GPU-only factorizations
●	●	●	Multicore and multi-GPU support
● NEW			MAGMA Analytics/MagmaDNN 0.2
●	●	●	LAPACK testing
●	●	●	Linux
●	●		Windows
●	●		Mac OS

MagmaDNN – Data Analytics Tool

- **MagmaDNN 0.2 – HP Data analytics and ML**
  GPU-accelerated numerical software using MAGMA as computational backend to accelerate its LA computations

- **Open source; looking for feedback and contributions**
  Started with students from REU/RECSEM program
  https://bitbucket.org/icl/magmadnn

- **MagmaDNN 0.2 main functionalities**
  - Tensors and tensor operations
  - Deep learning primitives:
    Fully-connected layers, convolutional layers, pooling layers, activation layers, and output layers.
  - SGD back-propagation training
  - Established adapters for calling CuDNN
  - Winograd convolutions to accelerate CNNs
  - Mixed-precision (FP16-FP32) FFT
  - Hyperparameter optimization framework
  - MNIST and CIFAR-10 benchmarks using MagmaDNN
  - Performance comparisons, accuracy validations, etc. (w\ TensorFlow, Theano, and PyTorch)

Provided in MAGMA 2.5

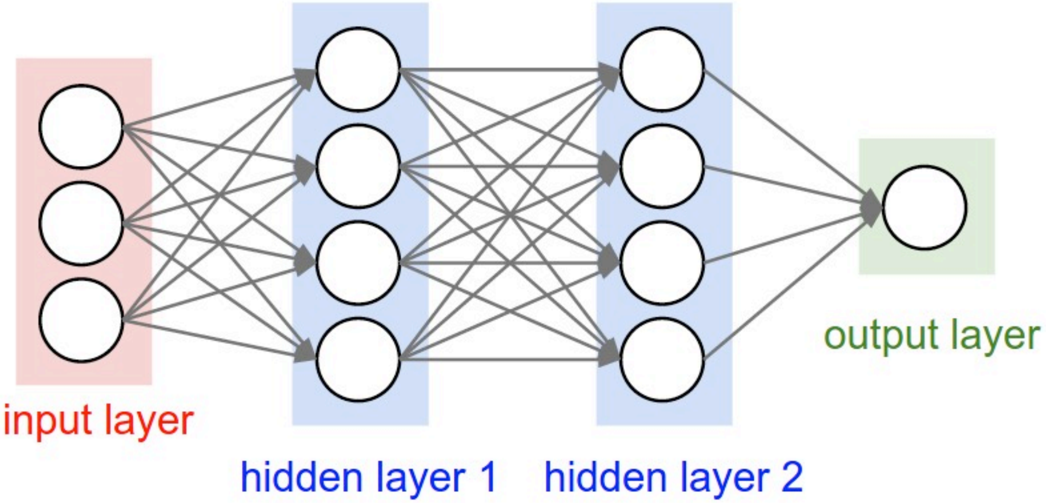
FEATURES AND SUPPORT

- **MAGMA 2.5** FOR CUDA
- **clMAGMA 1.4** FOR OpenCL
- **MAGMA MIC 1.4** FOR Intel Xeon Phi

CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●	●		Auxiliary BLAS
●			Batched LA
●		●	Sparse LA
●	●	●	CPU/GPU Interface
●	●	●	Multiple precision support
● NEW			Mixed precision (including FP16)
●			Non-GPU-resident factorizations
● NEW			GPU-only factorizations
●	●	●	Multicore and multi-GPU support
● NEW			MAGMA Analytics/MagmaDNN 0.2
●	●	●	LAPACK testing
●	●	●	Linux
●	●		Windows
●	●		Mac OS

Fully connected layers with MagmaDNN

Fully-connected 3-layer Neural Network example



- **Data** (input, output, NN weights, etc.) **is** handled through **tensor abstractions**

```
// 2d tensor for n_images and n_features in the corresponding dimensions
Tensor<float> Images = Tensor<float>({n_images, n_features});
```

- **Support for various layers:**
  Fully connected (FCLayer), convolution, activation, flatten, pooling, input, output, etc. layers

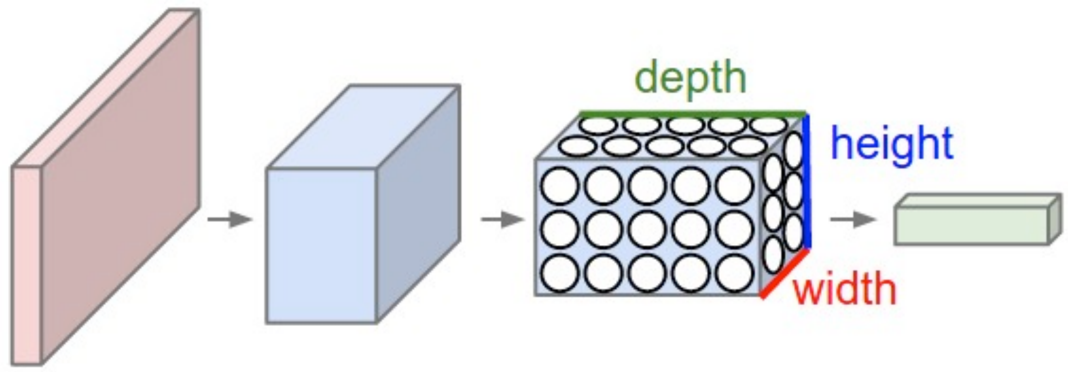
```
// Create layers for the network
FCLayer<float> *FC1             = new FCLayer<float>(&inputLayer, 128);
ActivationLayer<float> *actv1 = new ActivationLayer<float>(FC1, SIGMOID);
FCLayer<float> *FC2             = new FCLayer<float>(actv1, n_output_classes);
```

- **Support networks – composed of layers**

```
std::vector<Layer<float>*> vec_layer;
vec_layer.push_back(&inputLayer);
vec_layer.push_back(FC1);
vec_layer.push_back(actv1);
vec_layer.push_back(FC2);
…
```

Convolutional network layers

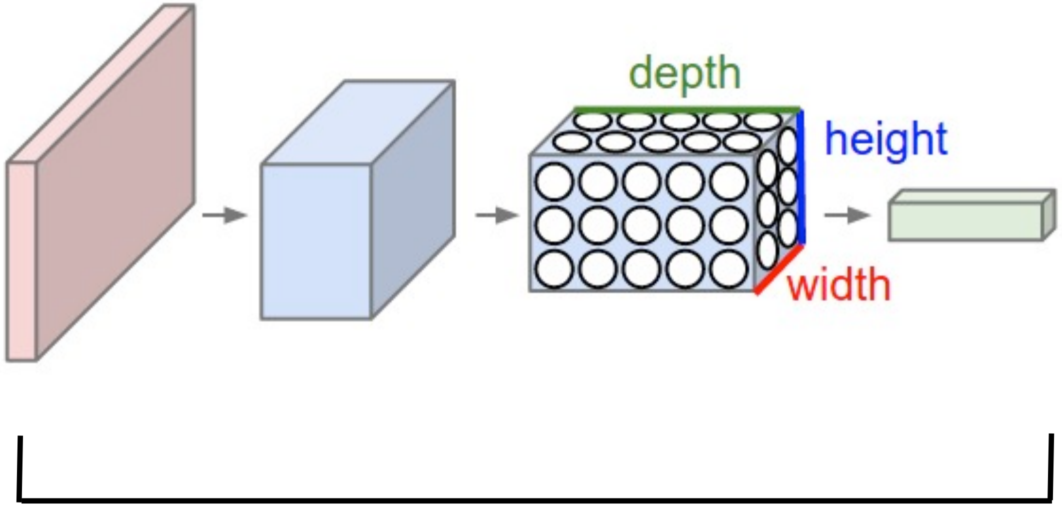
Convolution Network (ConvNet) example



- Layers are typically 3D volumes
- Handled through tensors
- Each layer transforms 3D tensor to 3D tensor
- Layers support the forward and backward pass algorithms for the training
- Support for optimization solvers (GD and derivatives)
  - Gradient Descent (GD)
  - Stochastic Gradient Descent (SGD)
  - Mini-Batch Gradient Descent (MB-GD)

How to accelerate on manycore GPU and CPUs?

Convolution Network (ConvNet) example



Require matrix-matrix products of various sizes, including batched GEMMs

- **Convolutions can be accelerated in various ways:**
  - **Unfold and GEMM**
  - **FFT**
  - **Winograd minimal filtering – reduction to batched GEMMs**

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

- **Use autotuning to handle complexity of tuning**

Accelerating CNNs in MagmaDNN with FFT

- **Convolutions $D_{i,c}$ * $G_{k,c}$** of images $D_{i,c}$ and filers $G_{k,c}$ can be **accelerated through FFT,** as shown by the following equality, consequence of the convolution theorem:

$$D_{i,c} * G_{k,c} = FFT^{-1} \left[FFT(D_{i,c}) .* FFT(G_{k,c}) \right],$$

  where .* is the Hadamard (component-wise) product, following the '.*' Matlab notation

- Developed **mixed-precision (FP16-FP32) FFT** using the GPU's Tensor Cores (TC) acceleration
  - **Dynamic splitting to increase the FP16 accuracy, while using high-performance TC**

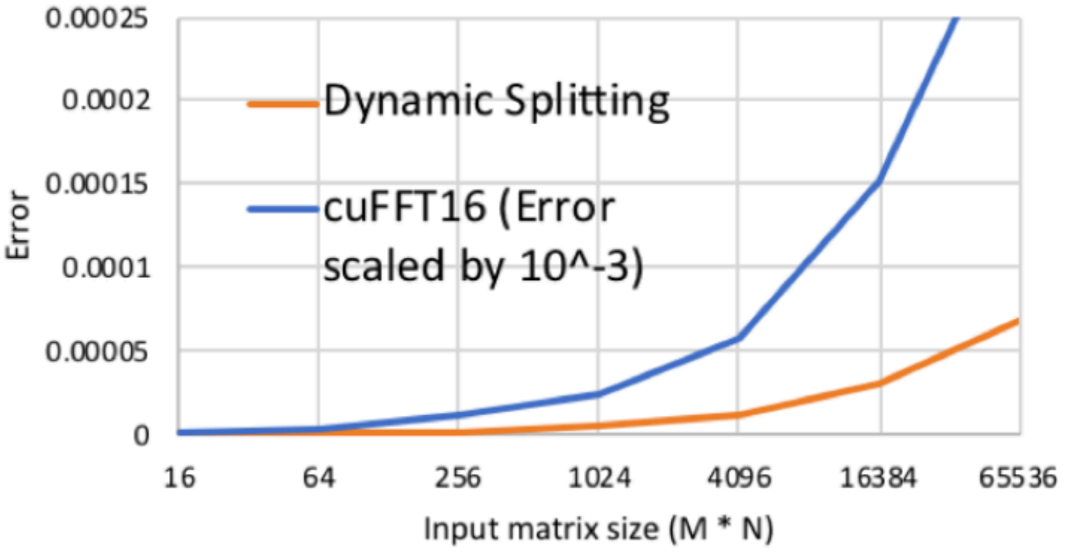
$$X_{FP32}(:) = s_1 \, X1_{FP16}(:) + s_2 \, X2_{FP16}(:)$$

[X1 X2] = FFT([X1 X2]   in FP16+ (e.g., go to radix 4, where the FFT matrix is exact in FP16)

$$FFT(X) \approx s_1 \, X1 + s_2 \, X2$$

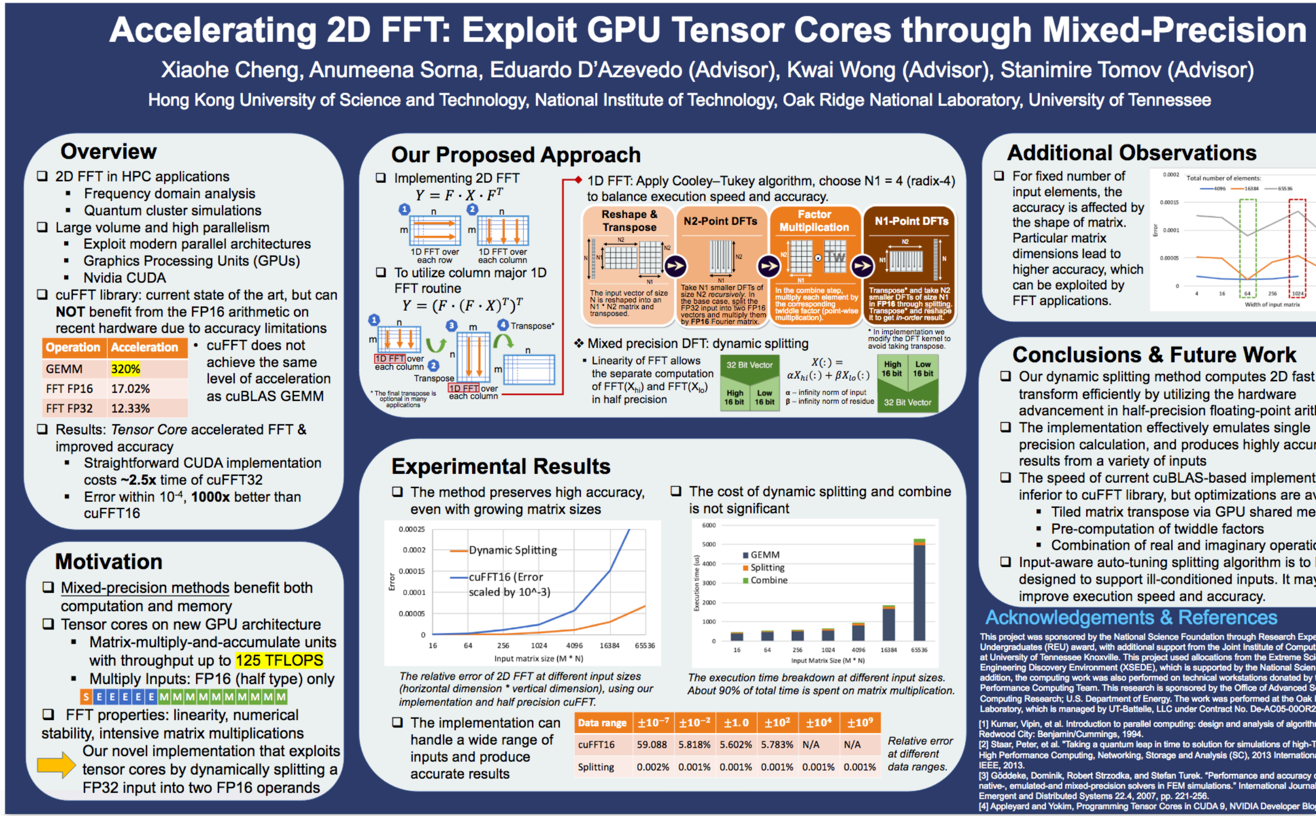
Accelerating CNNs with FFT

➢ **Accuracy of the mixed-precision (FP16-FP32) FFT**



Reference:
X. Cheng, A. Sorna , Ed D'Azevedo, K. Wong, S. Tomov, **"Accelerating 2D FFT: Exploit GPU Tensor Cores through Mixed-Precision,"** The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18), ACM Student Research Poster, Dallas, TX, November 11-16, 2018.

https://icl.utk.edu/projectsfiles/magma/pubs/77-mixed-precision-FFT.pdf
https://www.jics.utk.edu/recsem-reu/recsem18



Accelerating CNNs with Winograd's minimal filtering algorithm

- **FFT Convolution is fast for large filters;**
  Typical filters are small, e.g., 3x3, where Winograds's algorithm has been successful;
  In 2D, convolution of tile D of size 4x4 with filter F of size 3x3 is computed as

  $$D * F = A^T \, [\, [\, G\, D\, G^T] \;.\!*\; [B^T\, D\, B]\,]\, A$$

  where B, G, and A are given on the right:

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

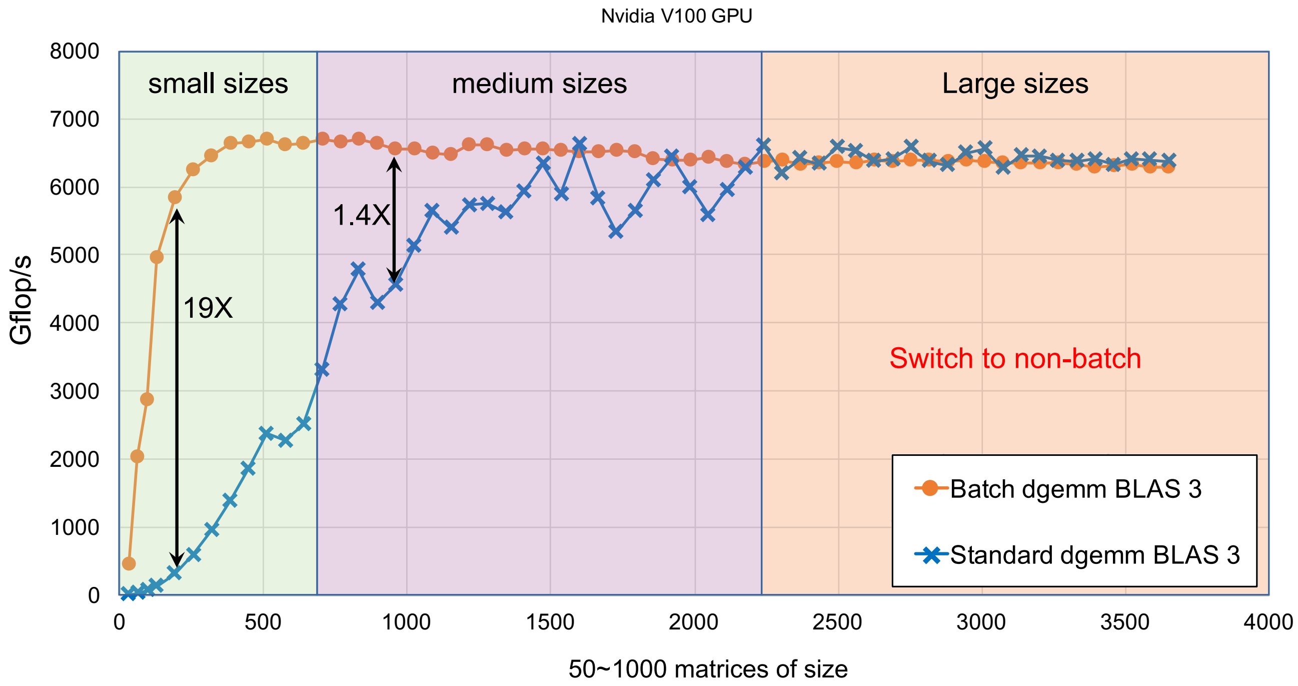
- Computing for a number of filters, sliding the tile over a batch of images, each with a number of channels, can be expressed as batched gemms, e.g.,

batch	m	n	k	
batch	**m**	**n**	**k**	**(sizes coming from VGG-16 CONVOLUTION LAYERS)**
16x64	12544	64	3	
16x64	**12544**	**64**	**64**	
16x16	12544	128	64	
16x16	**12544**	**128**	**128**	

…

How to implement fast batched DLA?

Problem sizes influence algorithms & optimization techniques

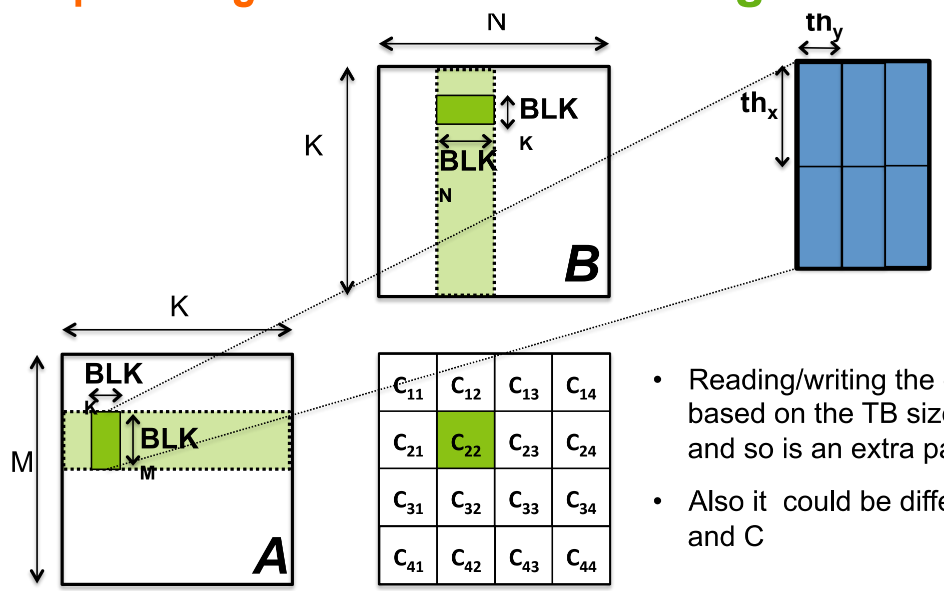


Matrix sizes (fixed) in the batch

Batch size 1,000 | Batch size 300 | Batch size 50

Kernels are designed various scenarios and parameterized for autotuning framework to find "best" performing kernels

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

Hyperparameter optimization framework

- **Hyperparameters are grouped in Model class**

```
// put in layers a sequence of predefined layers
std::vector<Layer<float>*> layers { &input_layer, FC1, actv1, FC2, output_layer };
// set some hyperparameters
Param p { learning_rate, weight_decay, batch_size, epochs };
Model model (p, &layers);

// train network model – arguments train data, train outcomes, verbose, accuracy, loss
model.fit(x_train, y_trian, false, accuracy, loss);
```

- **User can define a hyperparameter search space, e.g., start parameters, end, and step**

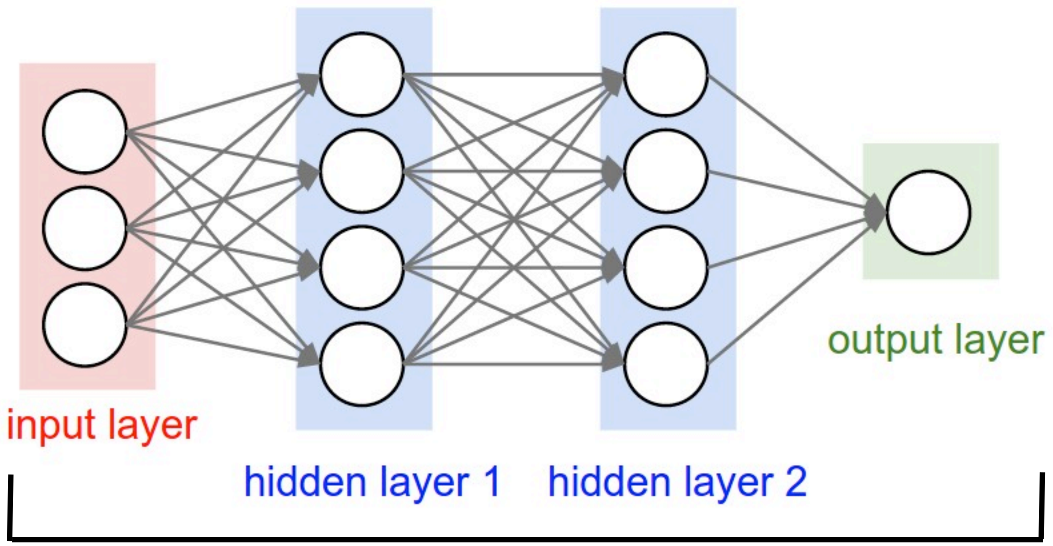
```
Param start  {  0.2,        0,  n_batch,       5 };
Param end    {  0.2,        1,  n_batch,       5 };
Param step   { 0.01,    0.01,         1,       1 };
Model model (start, &layers);
```

- **… and find optimal parameters via a grid_search function**

```
Param opt = grid_search(model, x_train, y_train, start, end, step, 5, -1, 5000, true);
```

MagmaDNN benchmarks and testing examples

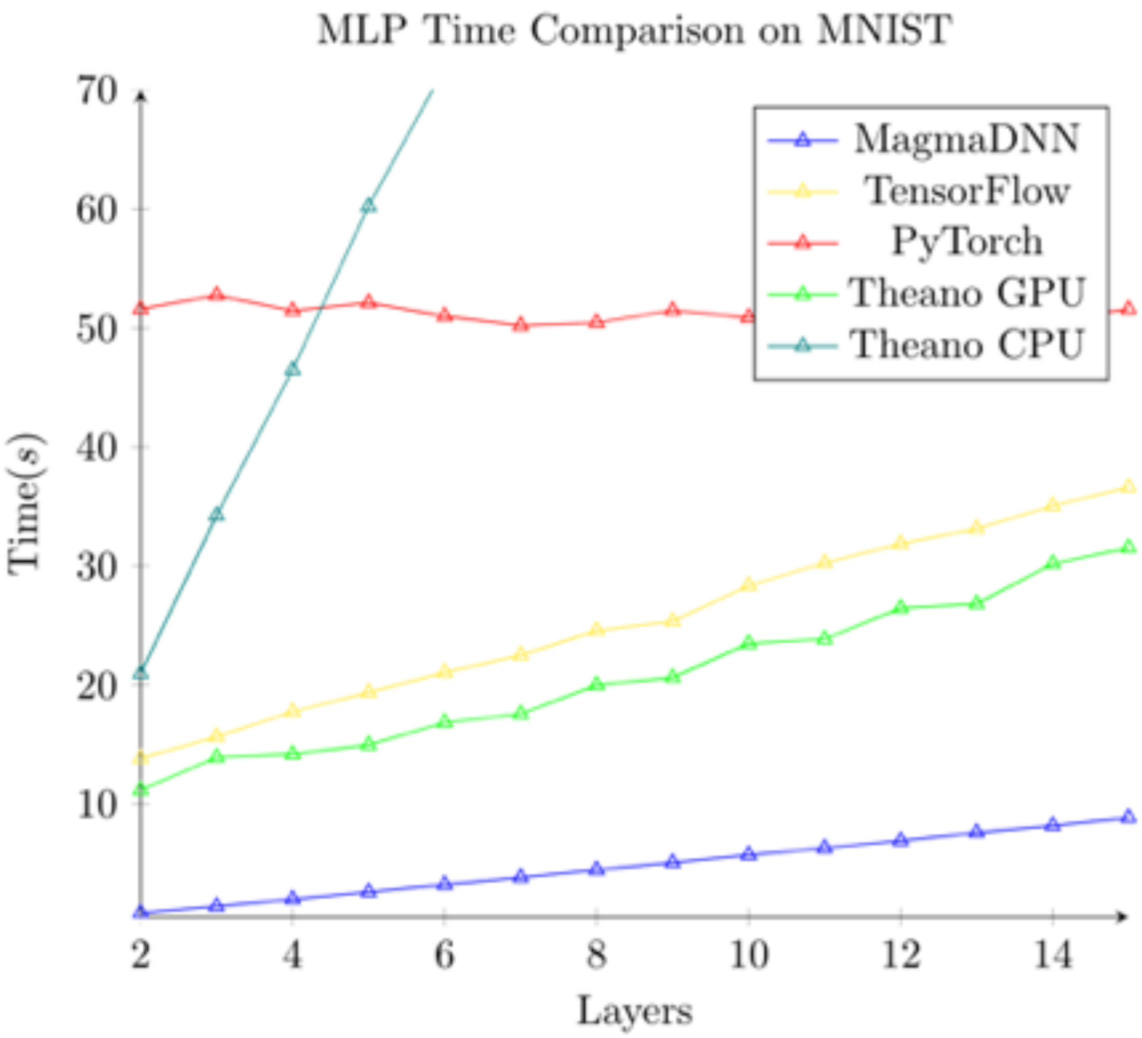
Fully-connected 3-layer Neural Network example



- The MNIST benchmark is a NN for recognizing handwritten numbers
- Input for the training are images of handwritten numbers and the labels indicating what are the numbers

- **MagmaDNN has testing/example drivers**
- **Example implementing the MNIST benchmark using MagmaDNN multilayer perceptron or a convolutional neural network**
- **CIFAR-10 benchmark using MagmaDNN**
- **Benchmarks for Wingrad and FFT**
- **Performance comparisons, accuracy validations, etc. (w\ TensorFlow, Theano, and PyTorch)**

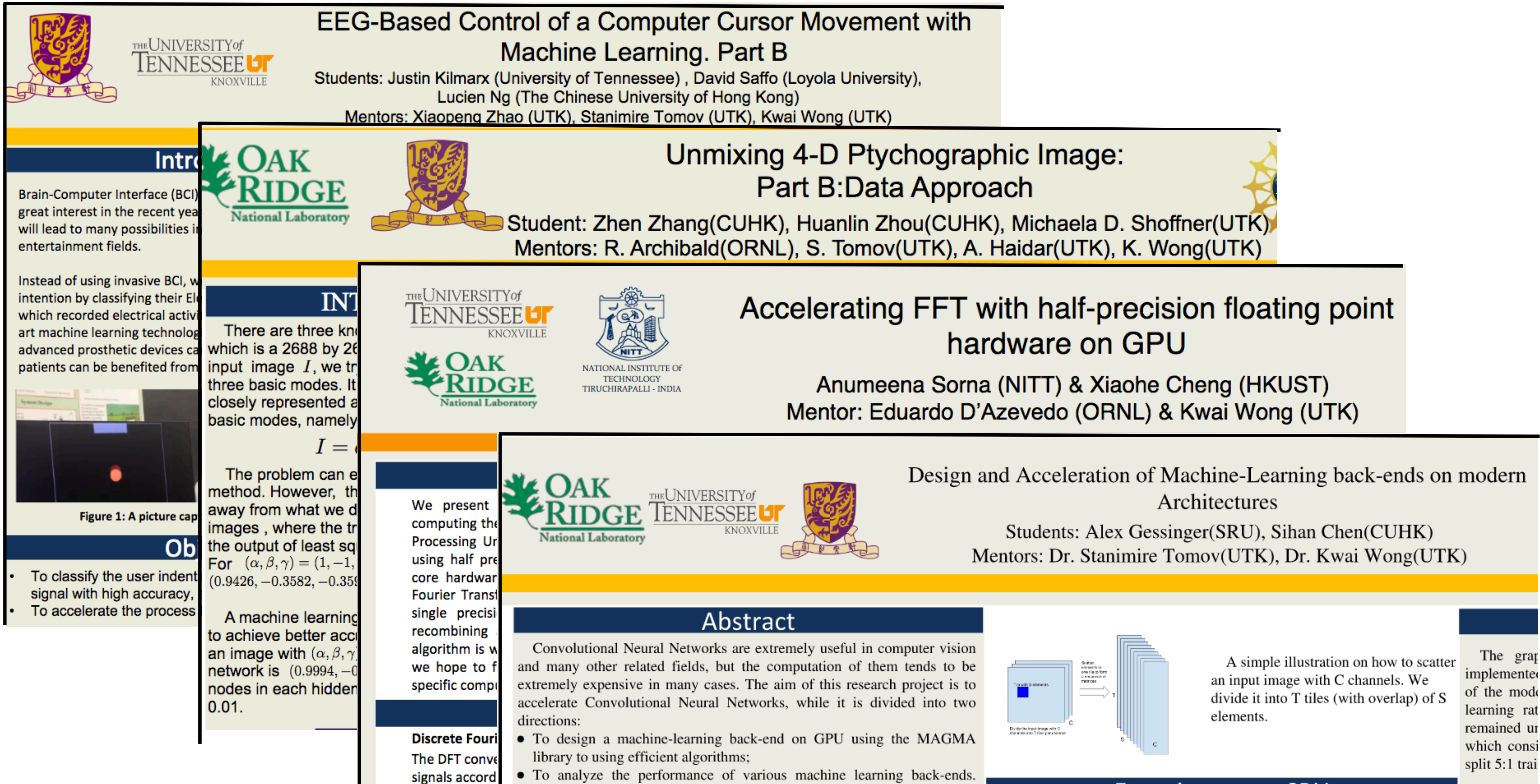
MagmaDNN performance benchmarks and validations



- MagmaDNN outperforms other popular deep learning libraries
- Compute time scales better than other libraries as models get larger

Parameter/Setting Name	Value
GPU	Nvidia 1050 Ti
CPU	Intel Xeon X5650 @ 2.67GHz x 12
OS	Ubuntu 16.04 LTS
Epochs	5
Batch Size	100
Learning Rate	0.2
Weight Decay	0.001
#Hidden Units / Layer	528

MagmaDNN benchmarks and testing examples …



Current work and Future directions

- **Performance portability and unified support on GPUs/CPUs**
  - C++ templates w/ polymorphic approach;
  - Parallel programming model based on CUDA, OpenMP task scheduling, and MAGMA APIs.
- **Hyperparameter optimization**
  - Critical for performance to provide optimizations that are application-specific;
  - A lot of work has been done (on certain BLAS kernels and the approach) but still need a simple framework to handle the entire library;
  - Current hyperparameter optimization tool must be further extended in functionalities
  - Add visualization and OpenDIEL to support ease of GPU deployment over large scale heterogeneous systems
- **Extend functionality, kernel designs, and algorithmic variants**
  - BLAS, Batched BLAS, architecture and energy-aware
  - New algorithms and building blocks, architecture and energy-aware
  - Randomization algorithms, e.g., for low-rank approximations, and applications
- **Use and integration with applications of interest (with ORNL collaborators)**
  - Brain-computer interface systems
  - Post-processing data from electron detectors for high-resolution microscopy studies (Unmixing 4-D Ptychographic Images)
  - Optimal cancer treatment strategies

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