

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/330132353>

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

Presentation · January 2019

DOI: 10.13140/RG.2.2.14906.64961

CITATIONS

0

READS

62

11 authors, including:



Lucien Ng

The Chinese University of Hong Kong

4 PUBLICATIONS 5 CITATIONS

SEE PROFILE



Alex Gessinger

Slippery Rock University of Pennsylvania

3 PUBLICATIONS 0 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Ginkgo - HPC Library for Sparse Linear Algebra [View project](#)



PLASMA [View project](#)

MagmaDNN 0.2

High-Performance Data Analytics for Manycore GPUs and CPUs

Lucien Ng¹, Sihan Chen¹, Alex Gessinger⁴, Daniel Nichols³, Sophia Cheng¹, Anu Meenasorna²

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 Tomov³, Azzam Haidar⁴, Ed D'Azevedo², 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)

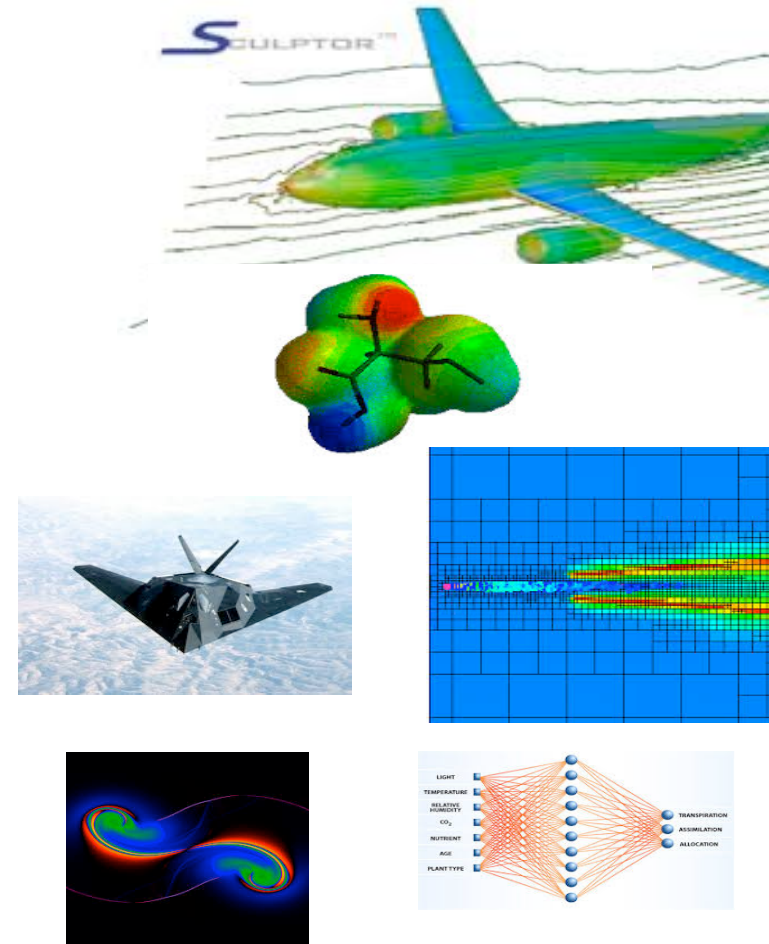
Research Experiences in Computational Science, Engineering, and Mathematics (RECSEM)

Knoxville, TN

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**

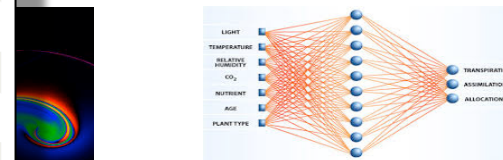
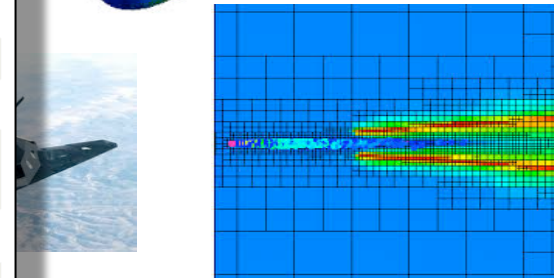
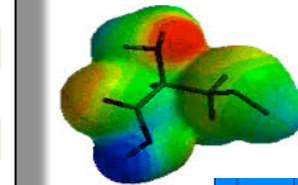
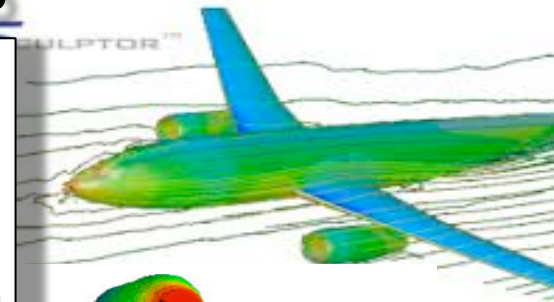
Provided in MAGMA 2.5

FEATURES AND SUPPORT

- ▶ **MAGMA 2.5** FOR **CUDA**
- ▶ **cMAGMA 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

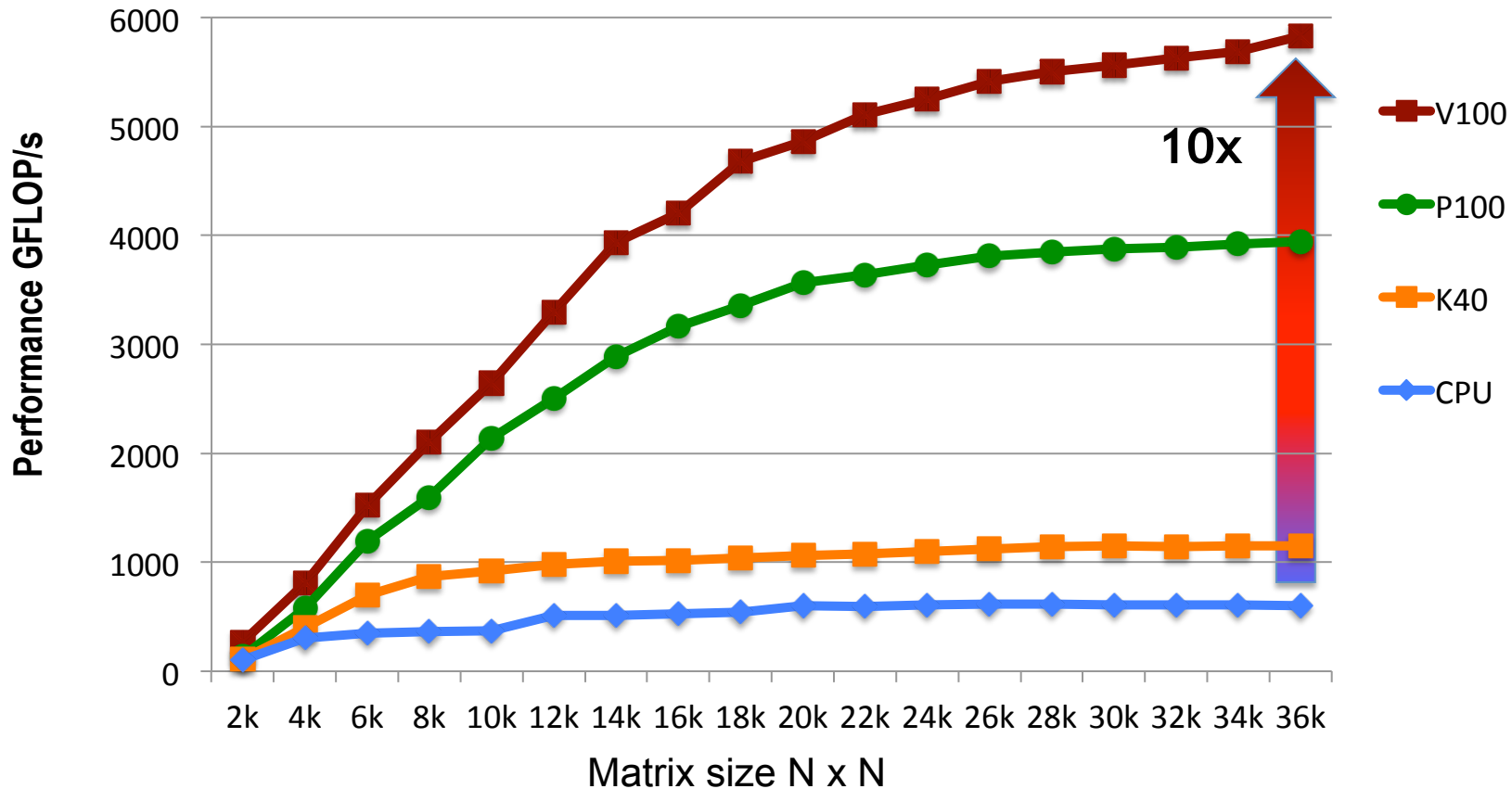
<http://icl.cs.utk.edu/magma>
<https://bitbucket.org/icl/magma>

Why use GPUs in HPC?

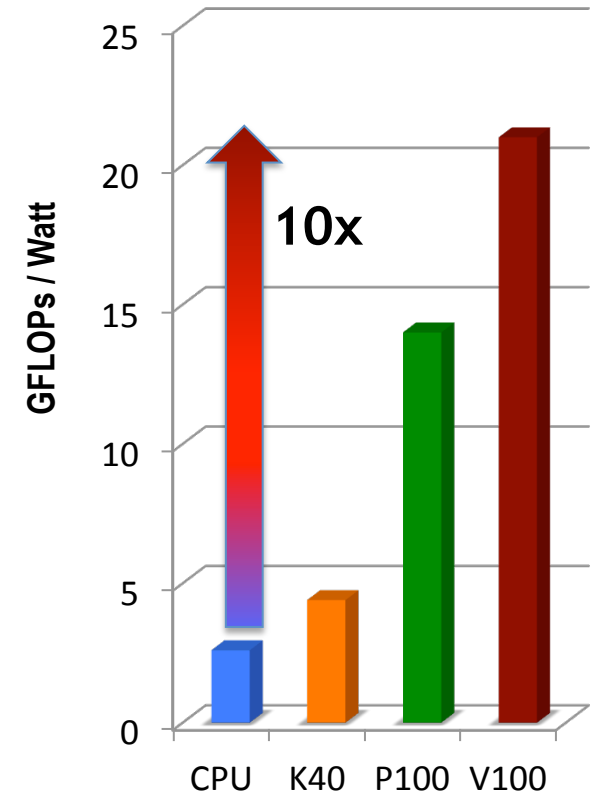
PERFORMANCE & ENERGY EFFICIENCY

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



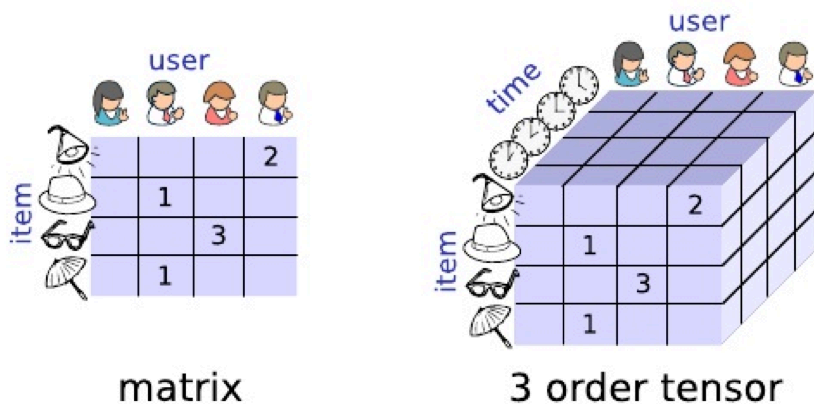
Energy efficiency (under ~ the same power draw)



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:

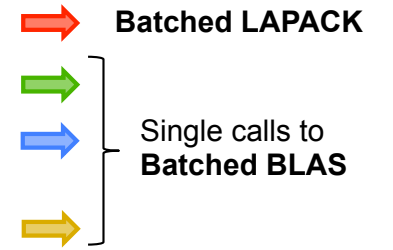
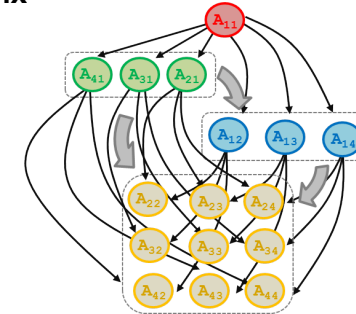
- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc.

Sparse/Dense solvers & preconditioners

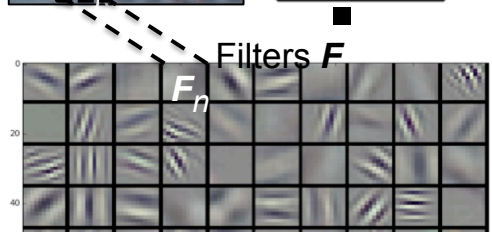
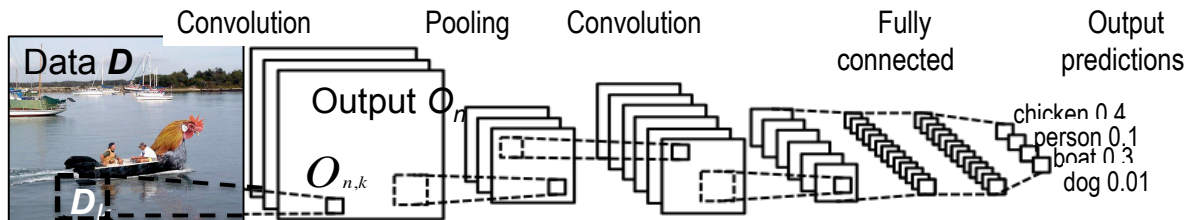
Sparse / Dense Matrix System

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

DAG-based factorization



Machine learning



Convolution of Filters F_n (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)

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:

$$\text{Batch}_{\{ C_{i3} = A^T B_{i3}, \text{ for range of } i3 \}}$$

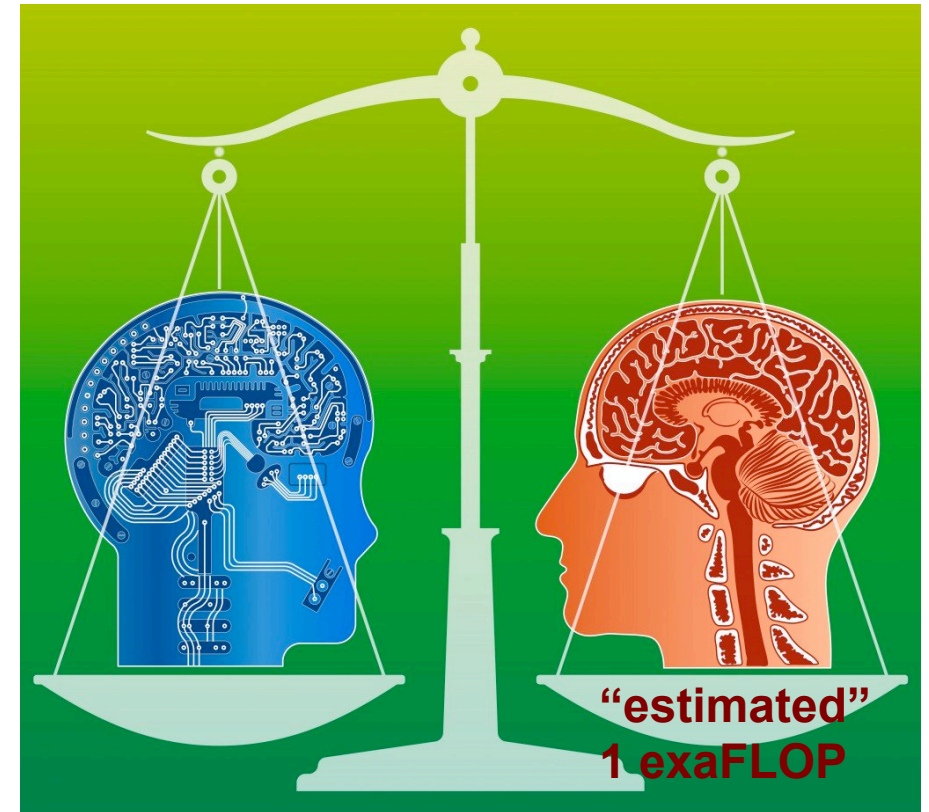
Machine learning / Artificial Intelligence

- 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
▶	CIMAGMA 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**

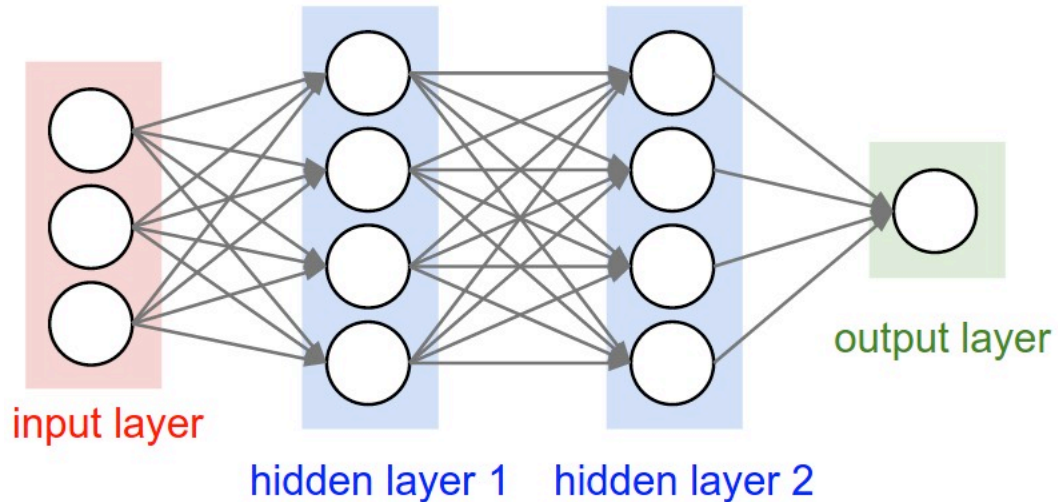
- Tensors and tensor operations
- Deep learning primitives:
Fully-connected layers, convolutional layers, pooling layers, activation layers, and output layers.
- SGD back-propagation training
- Established adapters for calling CuDNN
- Winograd convolutions to accelerate CNNs
- Mixed-precision (FP16-FP32) 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			
▶ CIMAGMA 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);
```

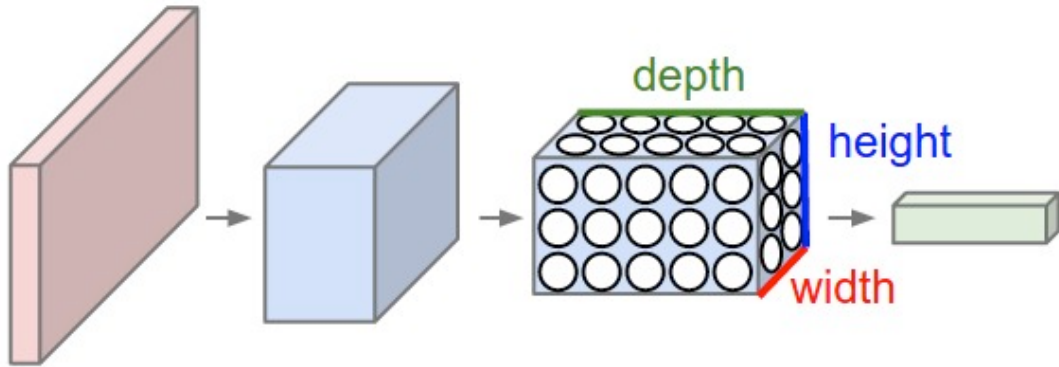
```
vec_layer.push_back(actv1);
```

```
vec_layer.push_back(FC2);
```

```
...
```

Convolutional network layers

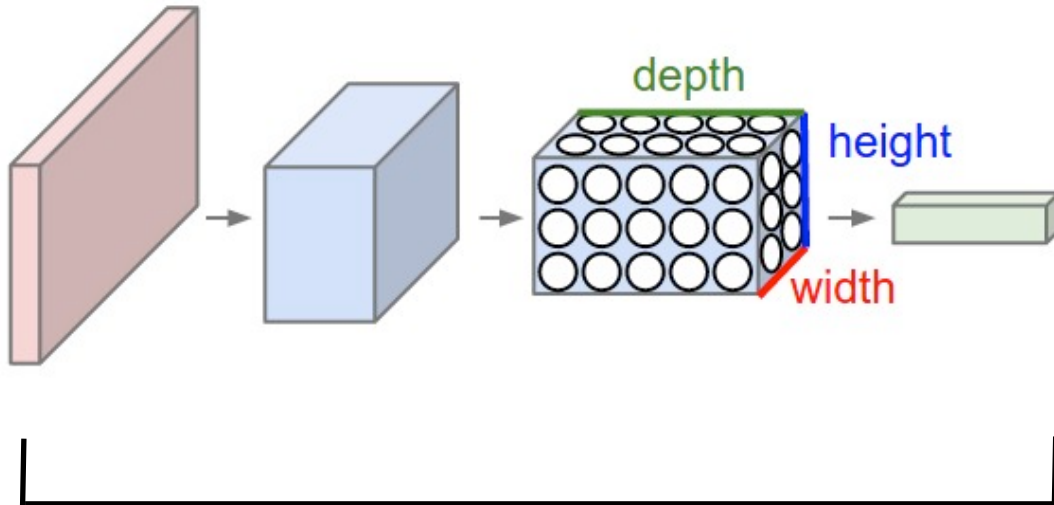
Convolution Network (ConvNet) example



- 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 filters $G_{k,c}$ can be **accelerated through FFT**, as shown by the following equality, consequence of the convolution theorem:

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

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**

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

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

$$\text{FFT}(X) \approx s_1 X1 + s_2 X2$$

Accelerating CNNs with FFT

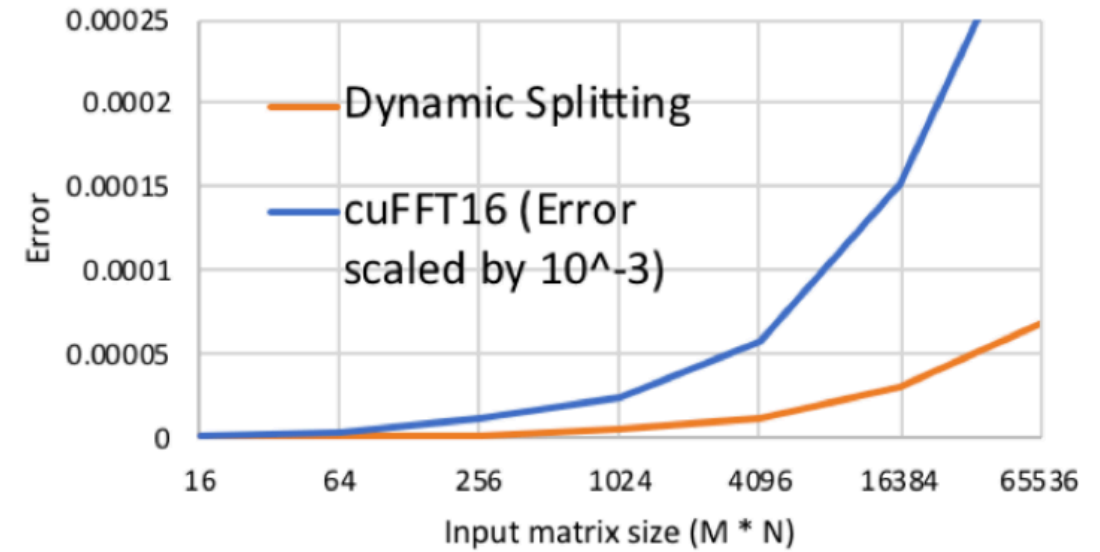
- Accuracy of the mixed-precision (FP16-FP32) FFT

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 2D FFT: Exploit GPU Tensor Cores through Mixed-Precision

Xiaohe Cheng, Anumeena Sorna, Eduardo D'Azevedo (Advisor), Kwai Wong (Advisor), Stanimire Tomov (Advisor)
Hong Kong University of Science and Technology, National Institute of Technology, Oak Ridge National Laboratory, University of Tennessee

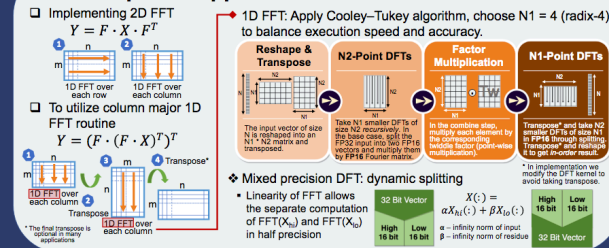
Overview

- 2D FFT in HPC applications
 - Frequency domain analysis
 - Quantum cluster simulations
- Large volume and high parallelism
 - Exploit modern parallel architectures
 - Graphics Processing Units (GPUs)
 - Nvidia CUDA
- cuFFT library: current state of the art, but can NOT benefit from the FP16 arithmetic on recent hardware due to accuracy limitations
 - cuFFT does not achieve the same level of acceleration as cuBLAS GEMM
- Results: Tensor Core accelerated FFT & improved accuracy
 - Straightforward CUDA implementation costs ~2.5x time of cuFFT32
 - Error within 10^{-4} , 1000x better than cuFFT16

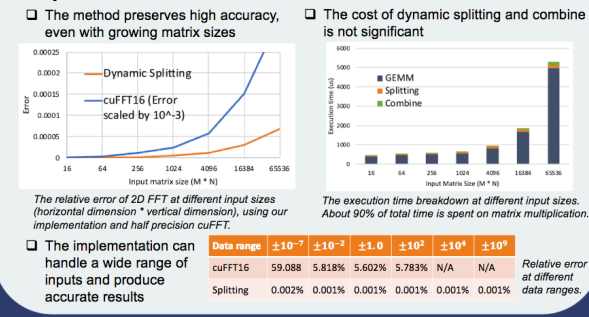
Motivation

- Mixed-precision methods benefit both computation and memory
- Tensor cores on new GPU architecture
 - Matrix-multiply-and-accumulate units with throughput up to **125 TFLOPS**
 - Multiply Inputs: FP16 (half type) only
- FFT properties: linearity, numerical stability, intensive matrix multiplications
 - Our novel implementation that exploits tensor cores by dynamically splitting a FP32 input into two FP16 operands

Our Proposed Approach



Experimental Results



Additional Observations

- For fixed number of input elements, the accuracy is affected by the shape of matrix. Particular matrix dimensions lead to higher accuracy, which can be exploited by FFT applications.

Conclusions & Future Work

- Our dynamic splitting method computes 2D fast transform efficiently by utilizing the hardware advancement in half-precision floating-point arithmetic.
- The implementation effectively emulates single precision calculation, and produces highly accurate results from a variety of inputs.
- The speed of current cuBLAS-based implementation is inferior to cuFFT library, but optimizations are available:
 - Tiled matrix transpose via GPU shared memory
 - Pre-computation of twiddle factors
 - Combination of real and imaginary operations
- Input-aware auto-tuning splitting algorithm is designed to support ill-conditioned inputs. It may improve execution speed and accuracy.

Acknowledgements & References

This project was sponsored by the National Science Foundation through Research Experience Undergraduates (REU) award, with additional support from the Joint Institute of Computational University of Tennessee Knoxville. This project used allocations from the Extreme Scale Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation. The computing work was also performed on technical workstations donated by the Performance Computing Team. This research is sponsored by the Office of Advanced Scientific Computing Research, U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR21400.

[1] Kumar, Vipin, et al. Introduction to parallel computing: design and analysis of algorithms. Rockwood City: Benjamin Cummings, 1994.
 [2] Stear, Peter, et al. "Taking a quantum leap in time to solution for simulations of high-performance computing." International Journal of High Performance Computing, Networking, Storage and Analysis (SC), 2013 International IEEE, 2013.
 [3] Göddelske, Dominik, Robert Strzodka, and Stefan Turek. "Performance and accuracy of real-time, emulated and mixed-precision solvers in FEM simulations." International Journal of High Performance Computing, Networking, Storage and Analysis (SC), 2013 International IEEE, 2013.
 [4] Appleyard and Yokim, Programming Tensor Cores in CUDA 9, NVIDIA Developer

Accelerating CNNs with Winograd's minimal filtering algorithm

- **FFT Convolution is fast for large filters;**
Typical filters are small, e.g., 3x3, where Winograd'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** **(sizes coming from VGG-16 CONVOLUTION LAYERS)**

16x64 12544 64 3

16x64 **12544** **64** **64**

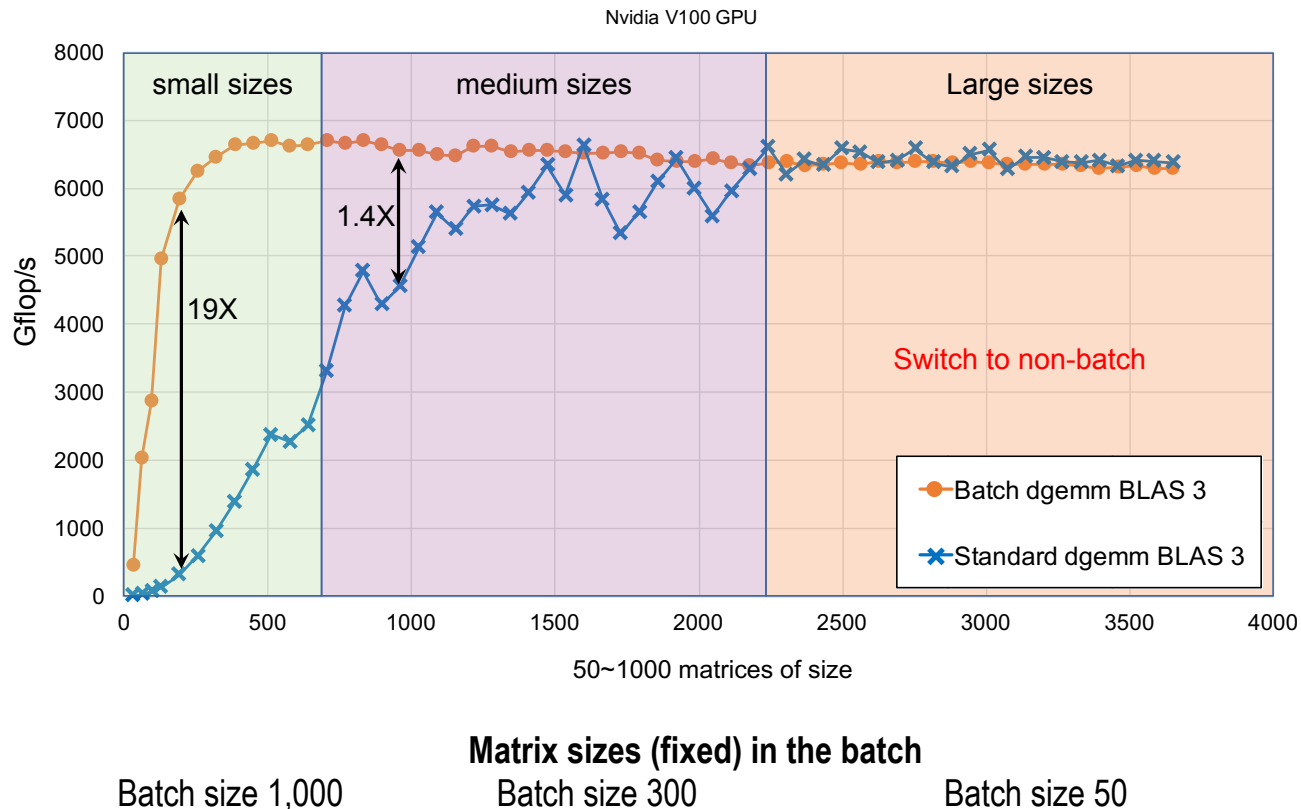
16x16 12544 128 64

16x16 **12544** **128** **128**

...

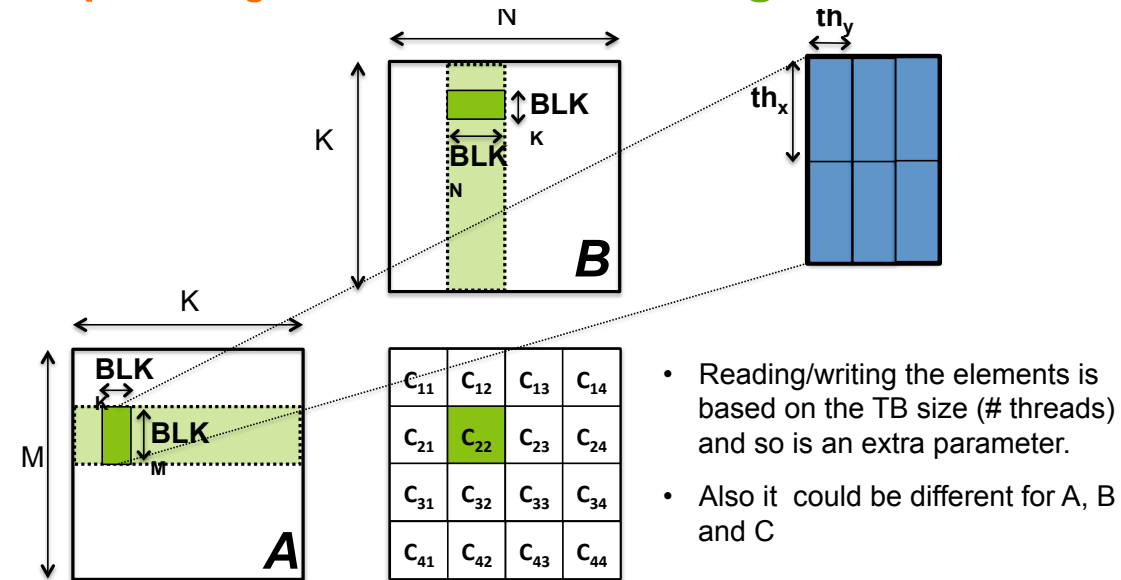
How to implement fast batched DLA?

Problem sizes influence algorithms & optimization techniques



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

Optimizing GEMM's: Kernel design



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_train, 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 };
```

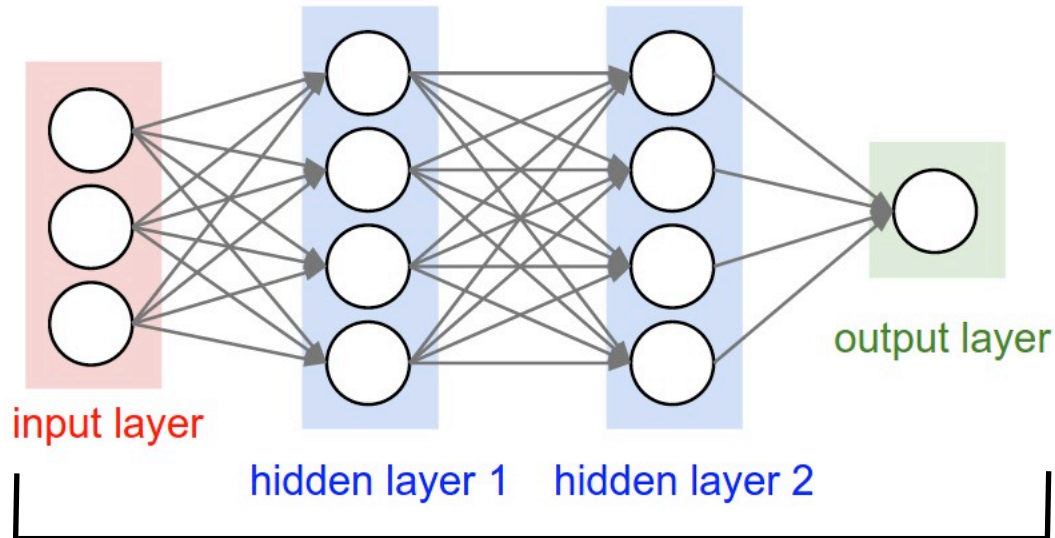
```
Model model (start, &layers);
```

- **... 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

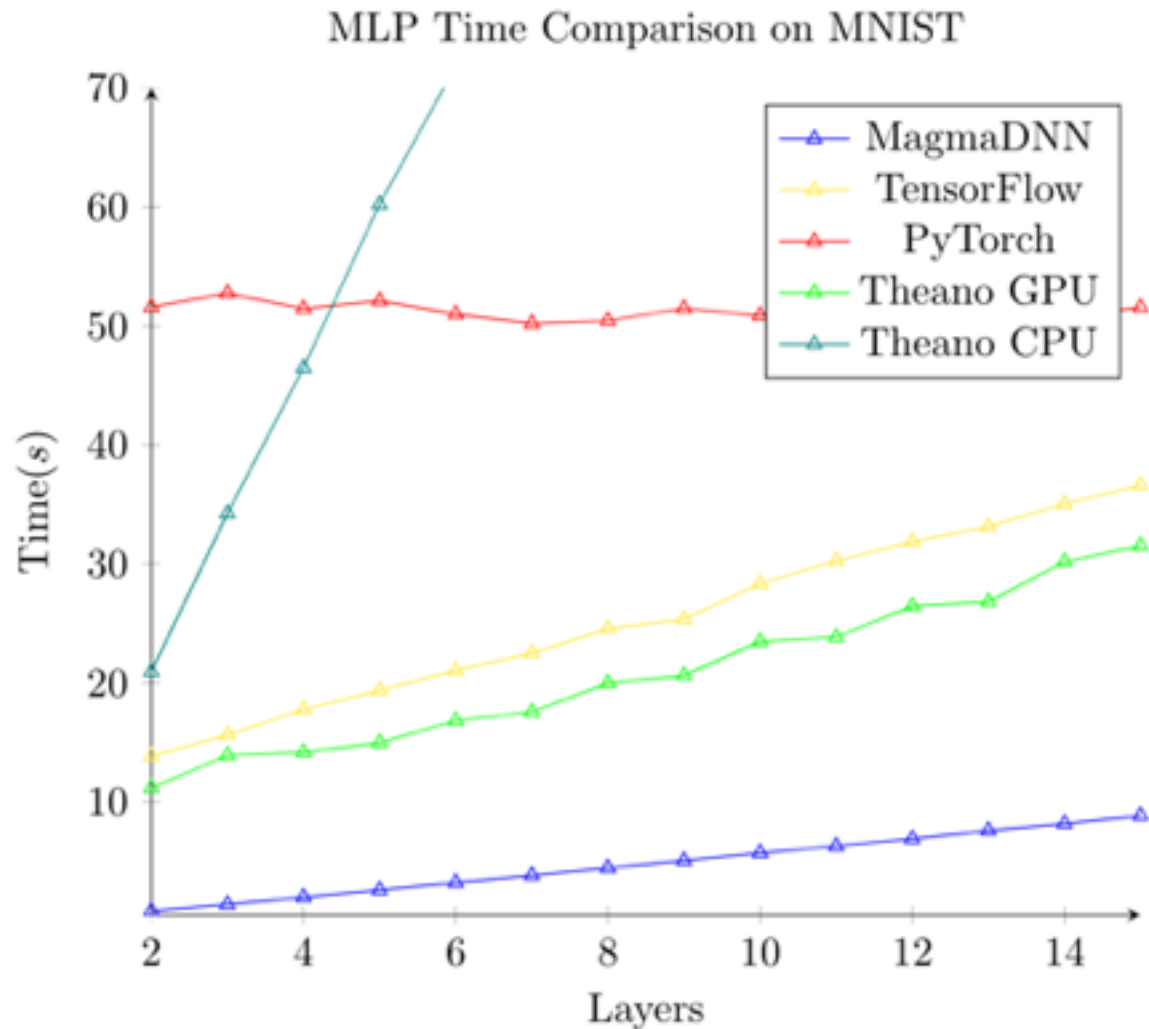
Fully-connected 3-layer Neural Network example



- 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 Winograd 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 ...



THE UNIVERSITY of TENNESSEE KNOXVILLE

EEG-Based Control of a Computer Cursor Movement with Machine Learning. Part B

Students: Justin Kilmarx (University of Tennessee) , David Saffo (Loyola University), Lucien Ng (The Chinese University of Hong Kong)
Mentors: Xiaopeng Zhao (UTK), Stanimire Tomov (UTK), Kwai Wong (UTK)

Intro

Brain-Computer Interface (BCI) has great interest in the recent years. It will lead to many possibilities in entertainment fields.

Instead of using invasive BCI, we can use non-invasive BCI. By using intention by classifying their EEG signals which recorded electrical activities of the brain. The advanced machine learning technology and advanced prosthetic devices can be benefited from BCI.






Figure 1: A picture capture of the BCI control interface.




Ob

- To classify the user intention signal with high accuracy,
- To accelerate the process

Unmixing 4-D Ptychographic Image: Part B:Data Approach

Student: Zhen Zhang(CUHK), Huanlin Zhou(CUHK), Michaela D. Shoffner(UTK)
Mentors: R. Archibald(ORNL), S. Tomov(UTK), A. Haidar(UTK), K. Wong(UTK)

Accelerating FFT with half-precision floating point hardware on GPU

Anumeena Sorna (NITT) & Xiaohe Cheng (HKUST)
Mentor: Eduardo D'Azevedo (ORNL) & Kwai Wong (UTK)




INT

There are three known methods for image inpainting which is a 2688 by 2688 input image I , we try to inpaint it in three basic modes. It is closely represented as a 2D image in three basic modes, namely

$$I = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}$$

The problem can be solved by the method. However, the method is far away from what we desire. For the output of least squares method, $(\alpha, \beta, \gamma) = (1, -1, 0.9426, -0.3582, -0.3582)$.

A machine learning method to achieve better accuracy. The network is a convolutional neural network with 100 nodes in each hidden layer and 10 nodes in the output layer.

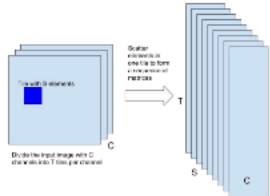
Design and Acceleration of Machine-Learning back-ends on modern Architectures

Students: Alex Gessinger(SRU), Sihan Chen(CUHK)
Mentors: Dr. Stanimire Tomov(UTK), Dr. Kwai Wong(UTK)

Abstract

Convolutional Neural Networks are extremely useful in computer vision and many other related fields, but the computation of them tends to be extremely expensive in many cases. The aim of this research project is to accelerate Convolutional Neural Networks, while it is divided into two directions:

- To design a machine-learning back-end on GPU using the MAGMA library to using efficient algorithms;
- To analyze the performance of various machine learning back-ends.



A simple illustration on how to scatter an input image with C channels. We divide it into T tiles (with overlap) of S elements.

The graph implementation of the modern machine learning remained unchanged which consists of a split 5:1 training and testing ratio.

Current work and Future directions

- **Performance portability and unified support on GPUs/CPU**
 - 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



U.S. DEPARTMENT OF
ENERGY

