MagmaDNN: Towards High-Performance Data Analytics and Machine Learning for Data-Driven Scientific Computing

D. Nichols[1], N.S. Tomov[1], F. Betancourt[1], **Stan Tomov**[1], K. Wang[1], J. Dongarra[1,2]

[1] Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

[2] Oak Ridge National Laboratory (ORNL), Oak Ridge

ISC High-Performance 2019
Workshop on "HPC Education and Training for Emerging Technologies" (HETET19)
Frankfurt, Germany
June 20, 2019

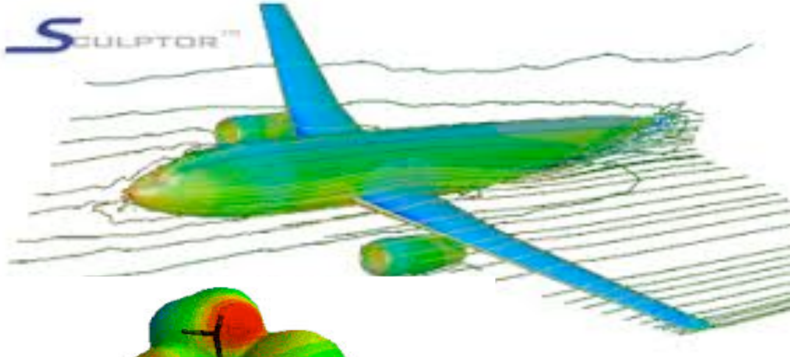
ICL INNOVATIVE COMPUTING LABORATORY

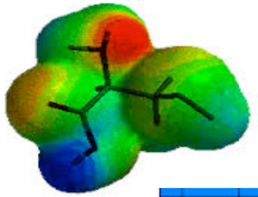
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

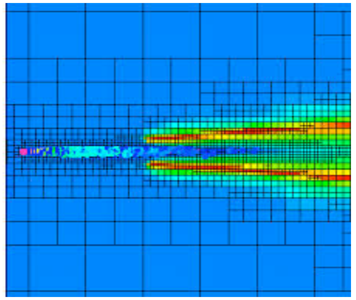
Dense Linear Algebra in Applications

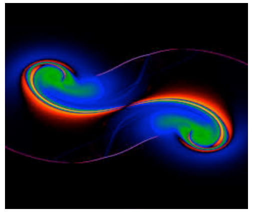
- **Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications, including ML and data analytics problems:**

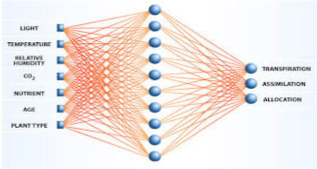
- **Linear systems:** **Solve $Ax = b$**
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** **Find x to minimize $\| Ax - b \|$**
  - Computational 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**











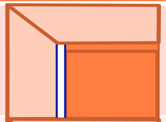
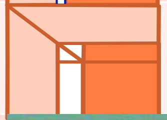
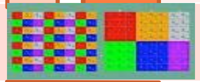
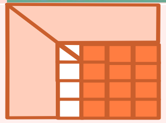
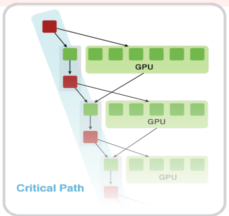
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

LA for modern architectures

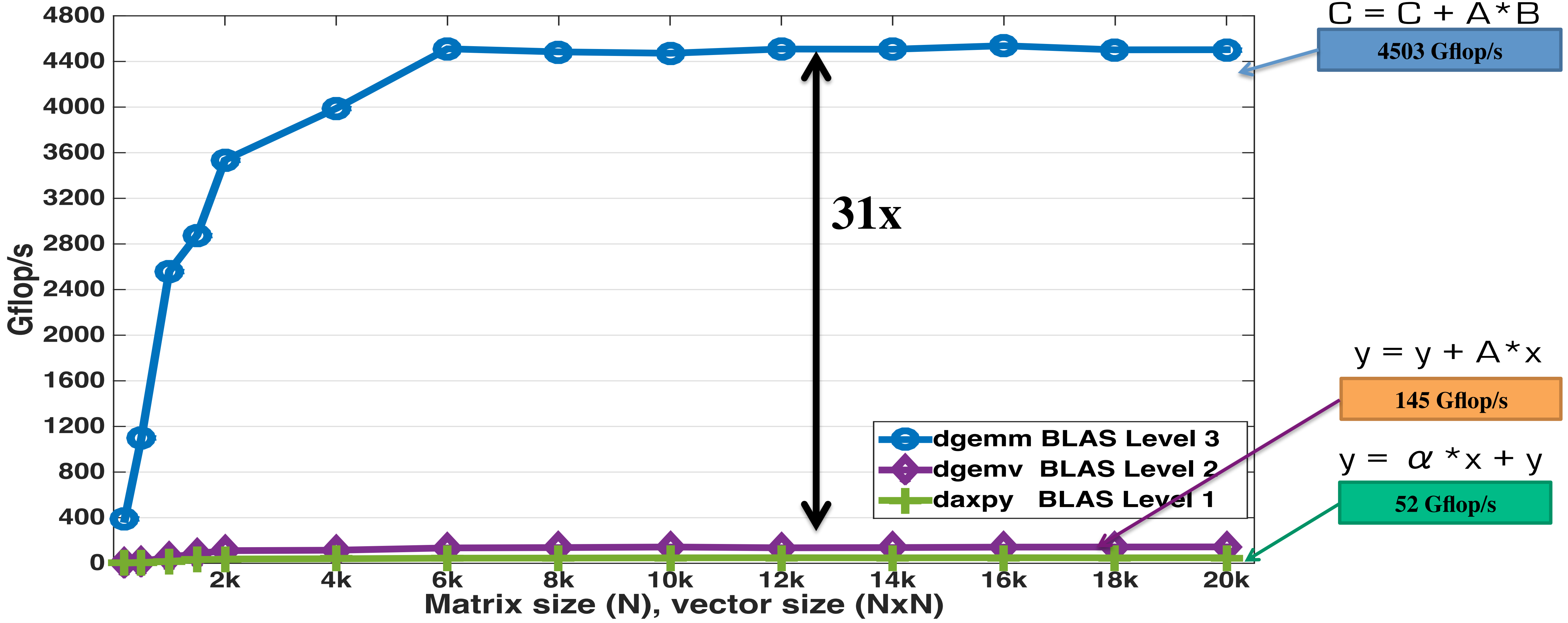
- **Leverage latest numerical algorithms and building blocks**
  MAGMA, PLASMA, SLATE (DOE funded) , MAGMA Sparse, POMPEI project*
- **Polymorphic approach**
  Use MAGMA sub-packages for various architectures;
  Provide portability through single templated sources using C++
- **Programming model**
  BLAS tasking + scheduling
- **Open standards**
  OpenMP4 tasking + MPI

Use of BLAS for portability

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Level 1 **BLAS**
LAPACK (80's) (Blocking, cache friendly)		Level 3 **BLAS**
ScaLAPACK (90's) (Distributed Memory)		**PBLAS**
PLASMA (00's) New Algorithms (many-core friendly)		**BLAS** on tiles + DAG scheduling
MAGMA Hybrid Algorithms (heterogeneity friendly)		**BLAS** tasking + (CPU / GPU / Xeon Phi) hybrid scheduling

Level 1, 2 and 3 BLAS

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



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

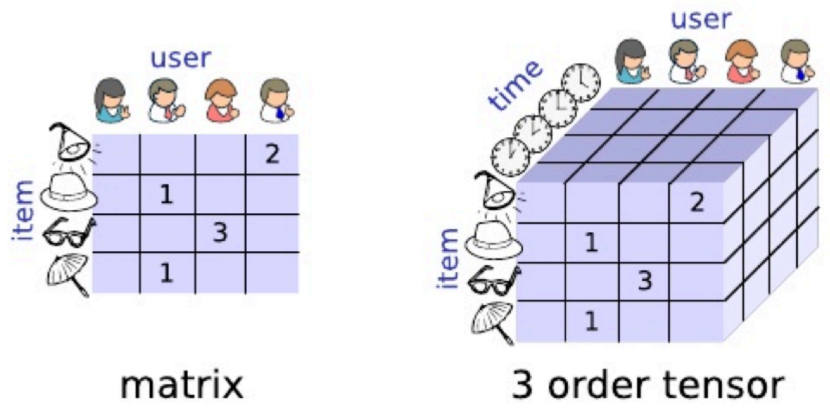
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

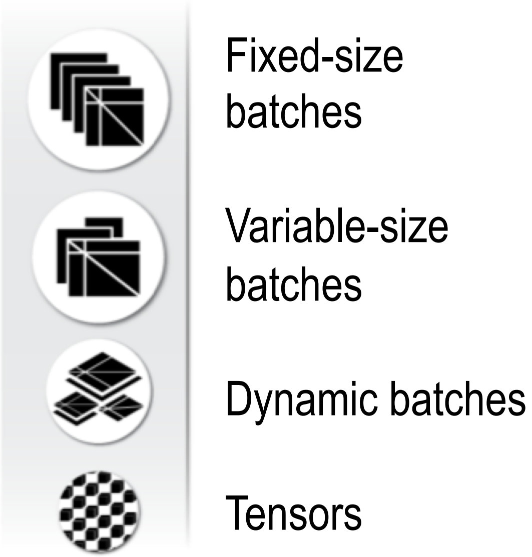
What about accelerated LA for Data Analytics?

- 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

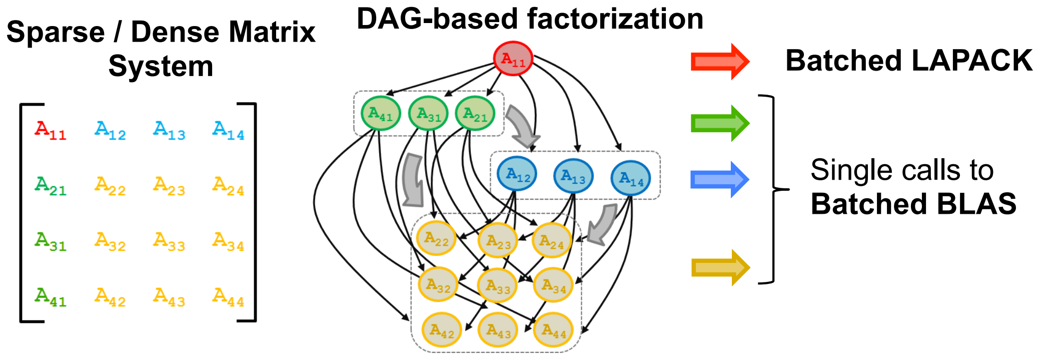


Data Analytics and LA on many small matrices

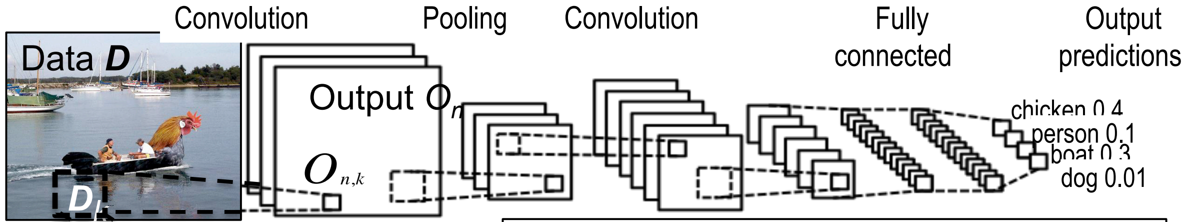
Data Analytics and associated with it Linear Algebra on small LA problems are needed in many applications:

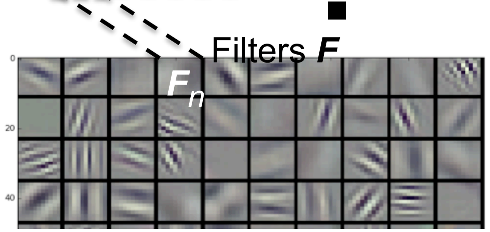
- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc.

Sparse/Dense solvers & preconditioners



Machine learning





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)

Applications using high-order FEM

- Matrix-free basis evaluation needs efficient tensor contractions,

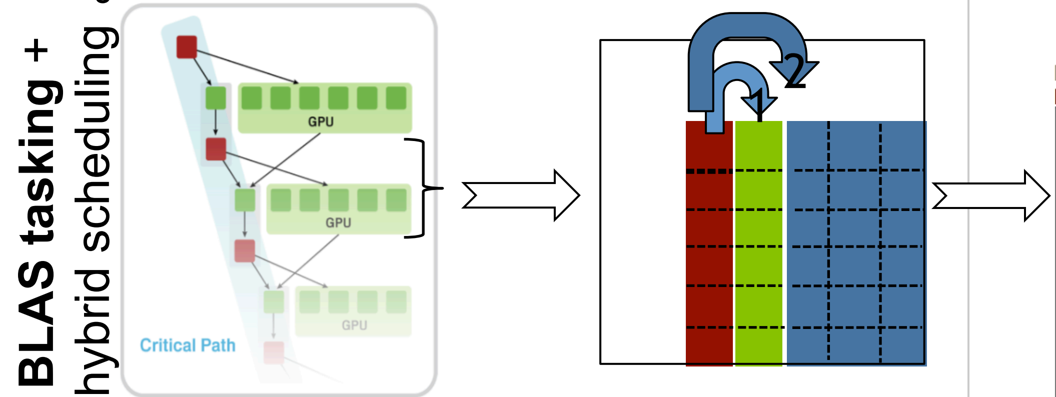
$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

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

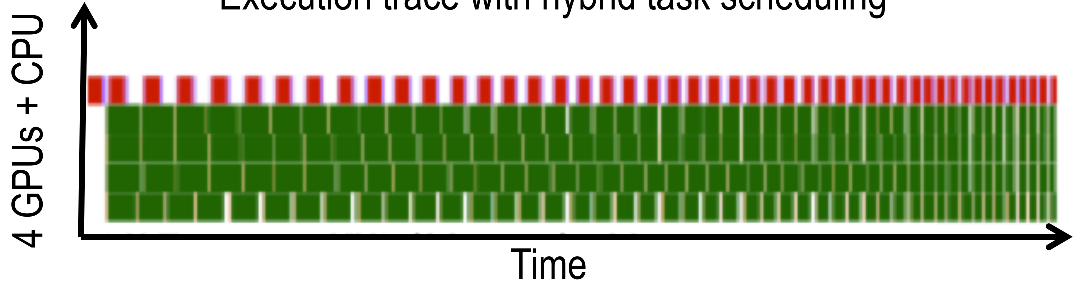
Batch_{ $C_{i3} = A^T B_{i3}$, for range of i3 }

Programming model: BLAS + scheduling

MAGMA
hybrid scheduling



Execution trace with hybrid task scheduling



MAGMA Dynamic

Left-looking hybrid Cholesky

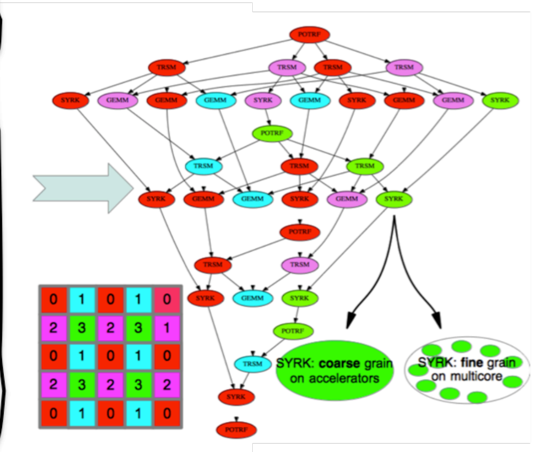
From sequential
LAPACK

```
for( j=0, j<n; j+=nb) {
  jb = min(nb, n-j);
  zherk( "Upper", "Conju
         &jb, &j, &one,

  if (j+jb < n)
    zgemm( "Conjugate
           dA(0,j), &ld

  zpotrf2( "Upper", &jb
  if (info != 0)
    *info += j;

  If (j+jb) < n) {

    ztrsm( "Left", "Uppe
           &jb, &n
  }
}
```

to parallel hybrid
MAGMA

```
for( j=0, j<n; j+=nb) {
  jb = min(nb, n-j);
  magma_zherk( MagmaUpper, MagmaConjTra
               jb, j, one, dA(0,j), ldda, one, dA
  magma_zgetmatrix_async( jb, jb, dA(j,j), ldd
  if (j+jb < n)
    magma_zgemm( MagmaConjTrans, Magma
                 dA(0,j), ldda, dA(0,j+jb), ldd
  magma_event_sync( event );
  zpotrf( MagmaUpperStr, &jb, work, &jb, info);
  if (info != 0)
    *info += j;
  magma_zsetmatrix_async(jb, jb, work, jb, dA
  If (j+jb) < n) {
    magma_event_sync( event );
    magma_ztrsm( MagmaLeft, MagmaUpper,
                 jb, n-j-jb, one, dA(j,j), ldda,
  }
}
```

MAGMA runtime environment



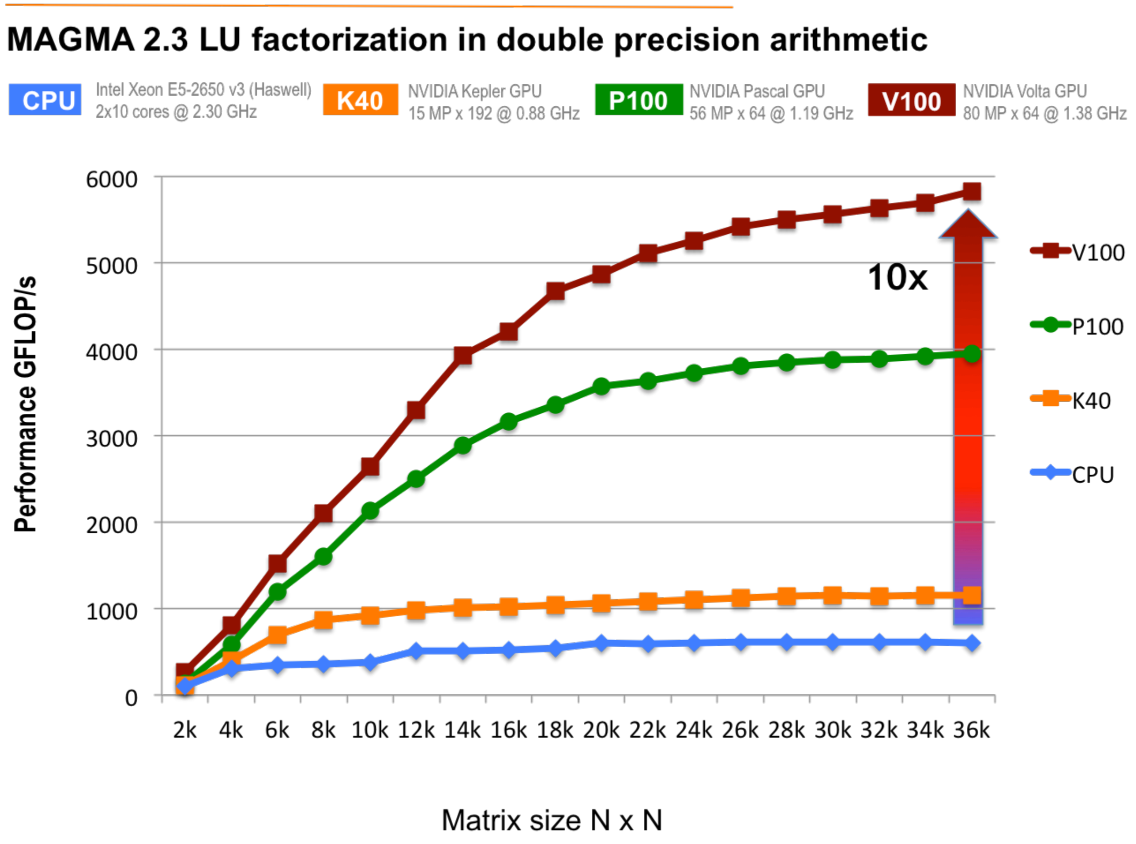
[A. Haidar, A. YarKhan, C. Cao, P. Luszczek, S. Tomov, and J. Dongarra, **"Flexible Linear Algebra Development and Scheduling with Cholesky Factorization"**, 17th IEEE International Conference on High Performance Computing and Communications, New York, August 2015.]

Note:
- MAGMA and LAPACK look similar
- Difference is lines in red, specifying data transfers and dependencies
- Differences are further hidden in a dynamic scheduler making the top level representation of MAGMA algorithms almost identical to LAPACK

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

Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code

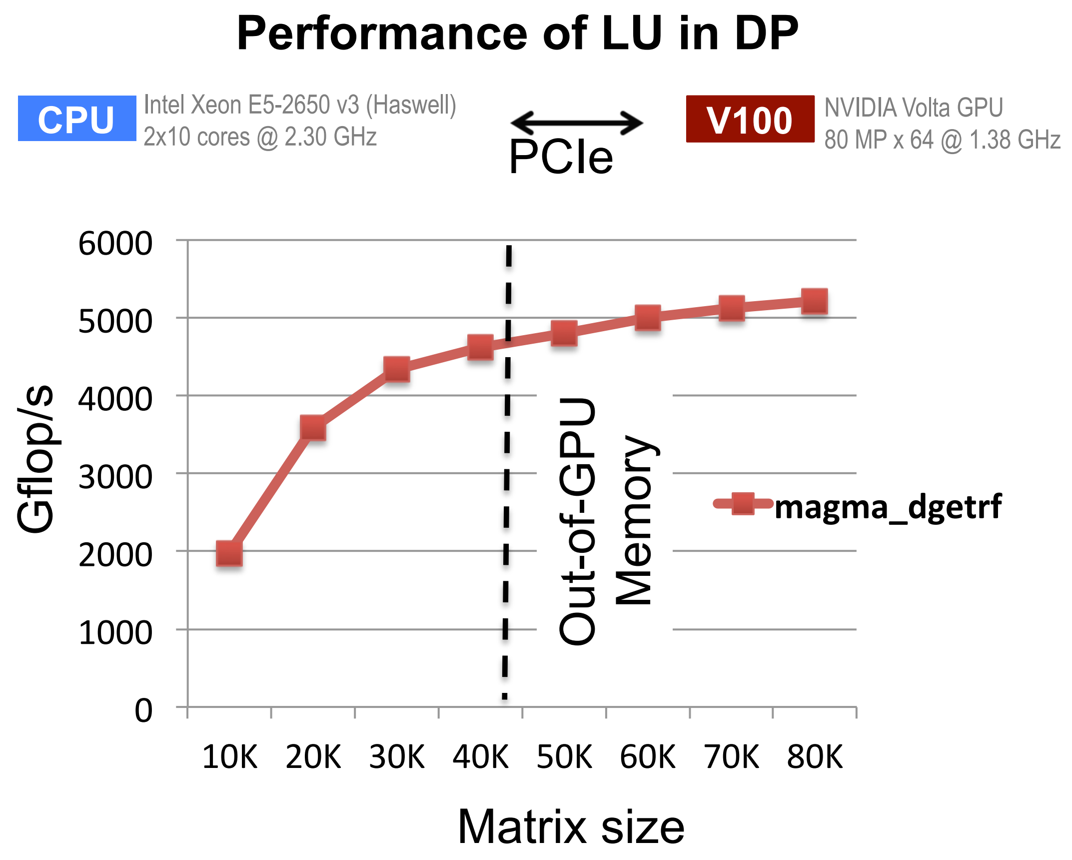


Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code
- Out-of-GPU memory algorithms
  - LA that is too large to fit into the main CPU/GPU memory

    A. Haidar, K. Kabir, D. Fayad, S. Tomov, and J. Dongarra, **"Out of Memory SVD Solver for Big Data"**, IEEE HPEC, September, 2017.

    Yuechao Lu, et al. on **out-of-GPU memory GEMMs in RSVD, TASMANIAN**, etc.



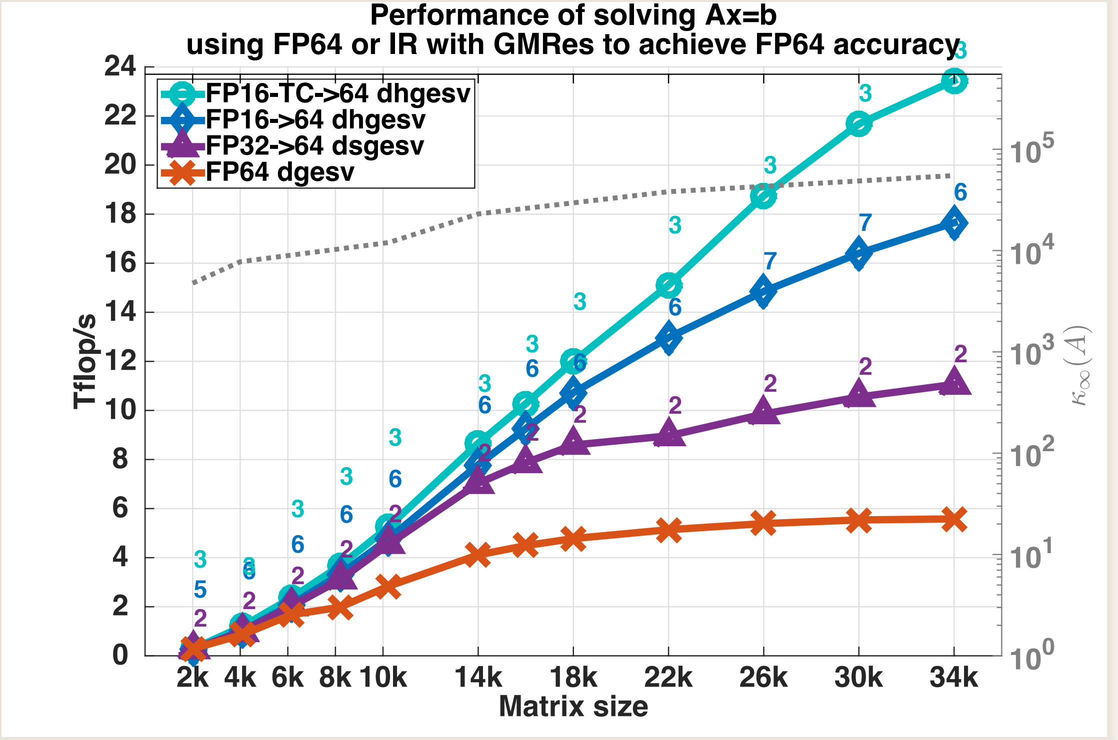
Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code
- Out-of-GPU memory algorithms
  - LA that is too large to fit into the main CPU/GPU memory
- Mixed-precision LA
  - Use new hardware features, e.g., Tensor Cores

    A. Haidar, P. Wu, S. Tomov, and J. Dongarra, **"Investigating half precision arithmetic to accelerate dense linear system solvers"**, SC'17 ScalA17 workshop, November 2017.

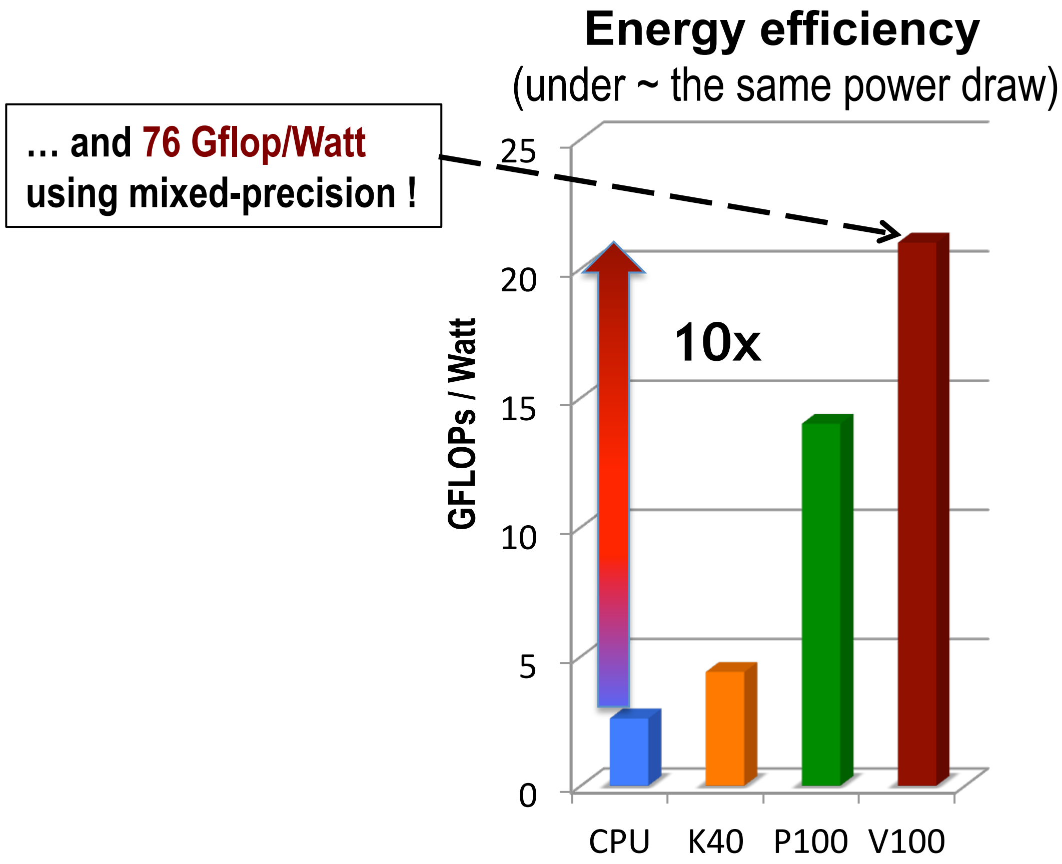
    A. Haidar, S. Tomov, and J. Dongarra, and N. Higham, "**Harnesing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers**", SC'18 (accepted), November 2018.

    Posters (GTC'18 2nd place, ISC'18 1st place; 11K downloads in a month)



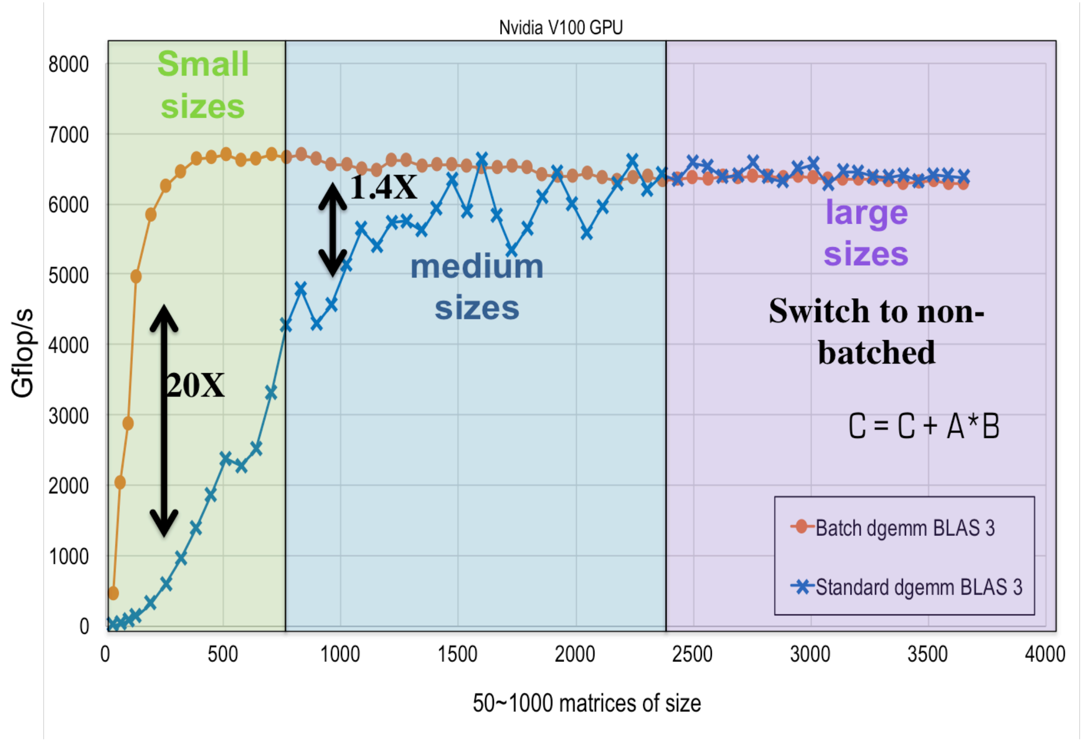
Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code
- Out-of-GPU memory algorithms
  - LA that is too large to fit into the main CPU/GPU memory
- Mixed-precision LA
  - Use new hardware features, e.g., Tensor Cores
- Energy efficient
  - Build energy awareness and tradeoff with performance



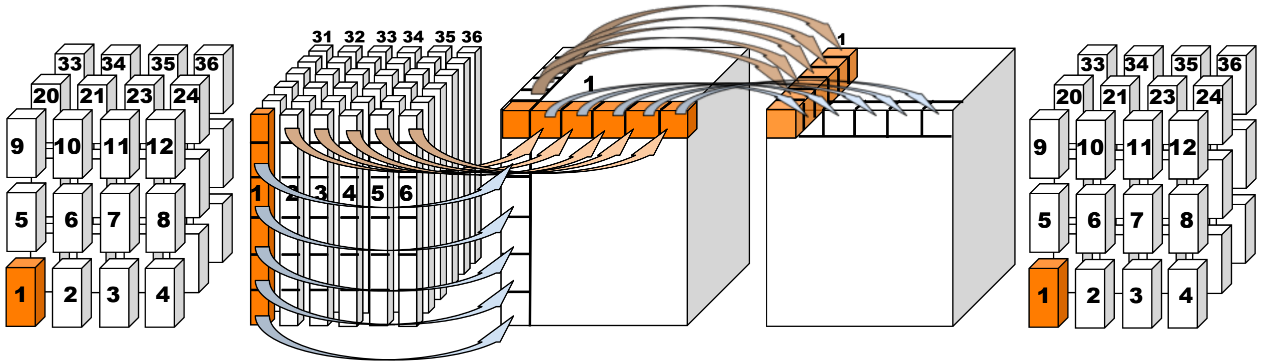
Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code
- Out-of-GPU memory algorithms
  - LA that is too large to fit into the main CPU/GPU memory
- Mixed-precision LA
  - Use new hardware features, e.g., Tensor Cores
- Energy efficient
  - Build energy awareness and tradeoff with performance
- Batched LA
  - LA on many small matrices

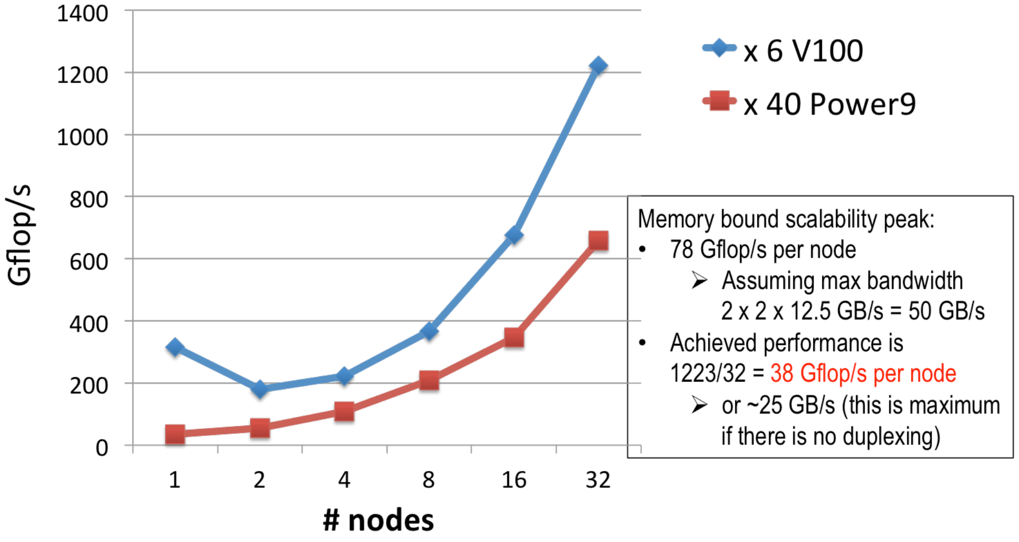


Main Classes of Algorithms in MAGMA

- Hybrid algorithms
  - Use both CPUs and GPUs
- GPU-only algorithms
  - Entirely GPU code
- Out-of-GPU memory algorithms
  - LA that is too large to fit into the main CPU/GPU memory
- Mixed-precision LA
  - Use new hardware features, e.g., Tensor Cores
- Energy efficient
  - Build energy awareness and tradeoff with performance
- Batched LA
  - LA on many small matrices
- FFT
  - FFTs, convolutions, auxiliary routines (transposes, matricizations, etc.)

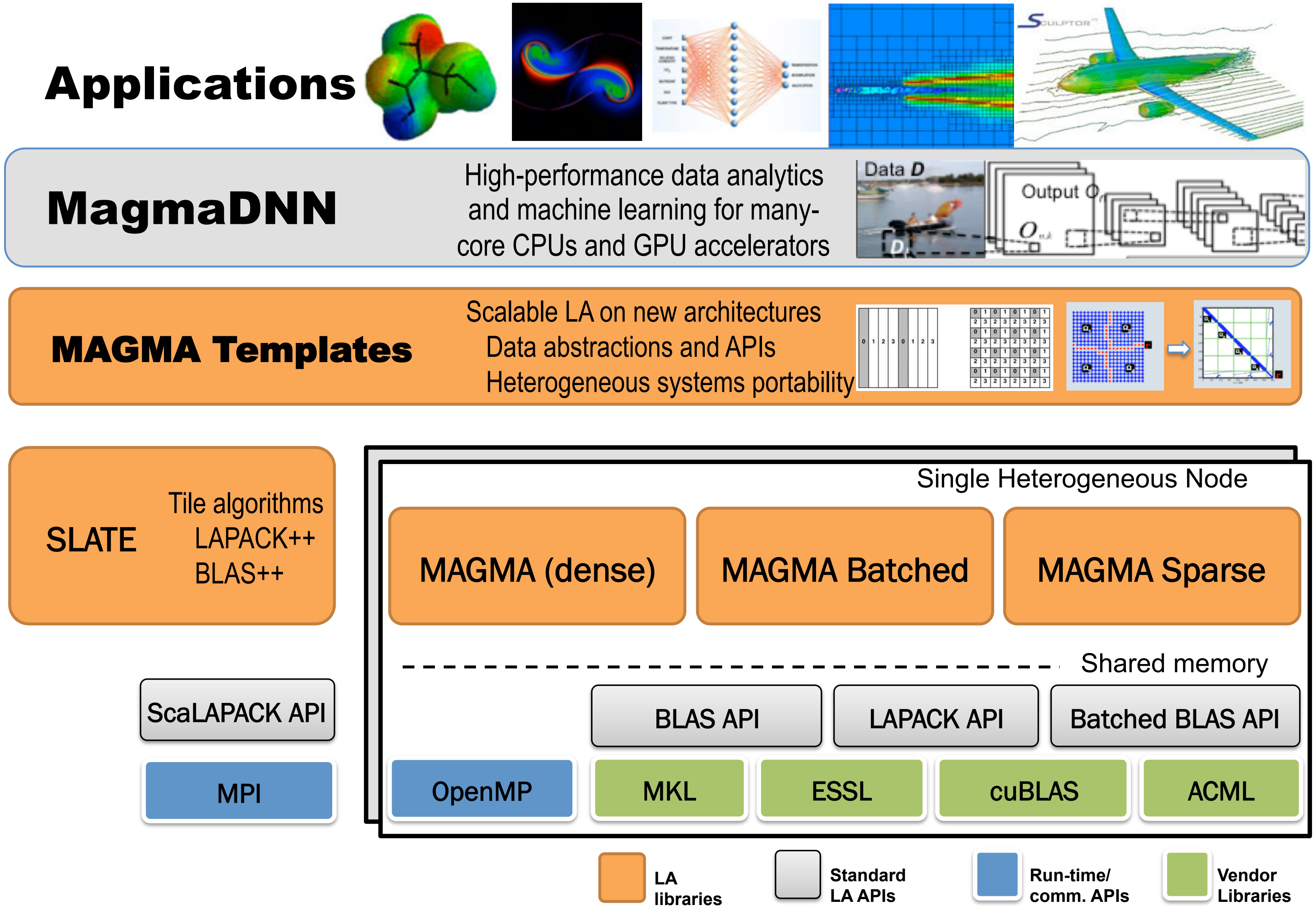


Strong scalability of 3D FFT on Summit (N = 1024)



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

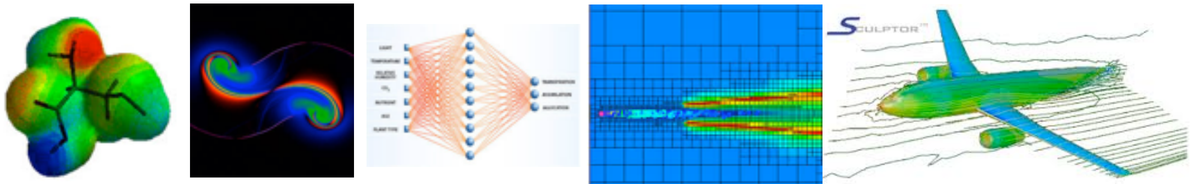
MagmaDNN



- **MagmaDNN is HP Data Analytics and ML framework built around the MAGMA libraries aimed at providing a modularized and efficient tool for training DNNs.**
- **MagmaDNN makes use of the highly optimized MAGMA libraries giving significant speed boosts over other modern frameworks.**

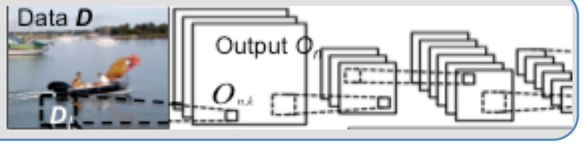
MagmaDNN

Applications



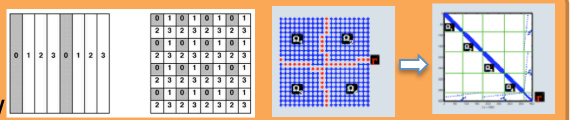
MagmaDNN

High-performance data analytics and machine learning for many-core CPUs and GPU accelerators



MAGMA Templates

Scalable LA on new architectures
Data abstractions and APIs
Heterogeneous systems portability



SLATE

Tile algorithms
LAPACK++
BLAS++

ScaLAPACK API

MPI

Single Heterogeneous Node

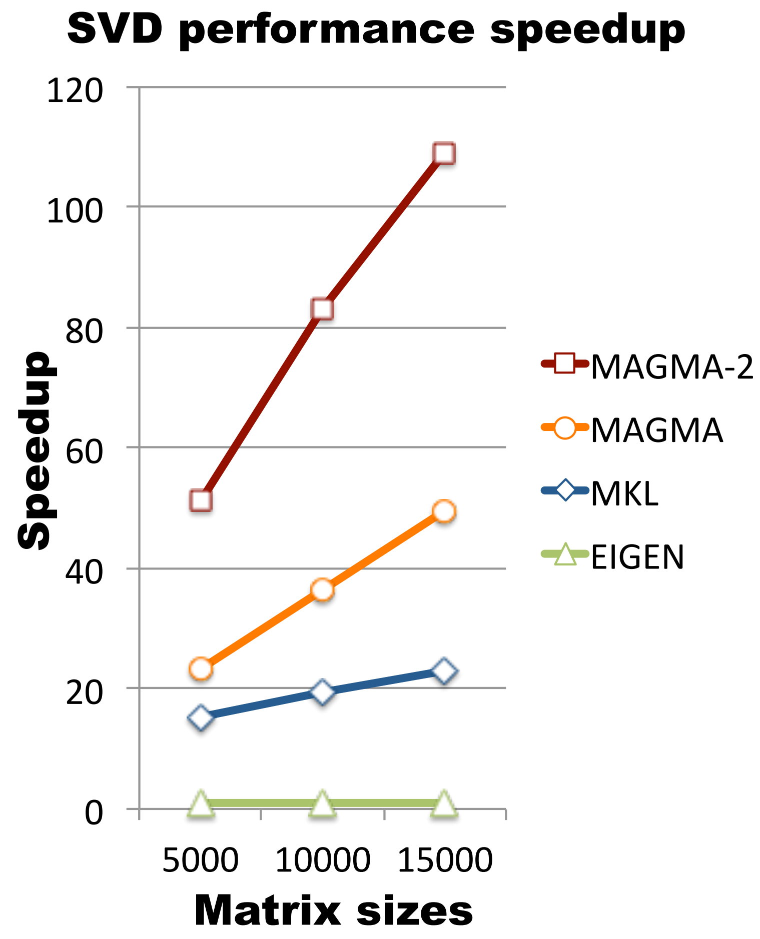
MAGMA (dense) | MAGMA Batched | MAGMA Sparse

Shared memory

BLAS API | LAPACK API | Batched BLAS API

OpenMP | MKL | ESSL | cuBLAS | ACML

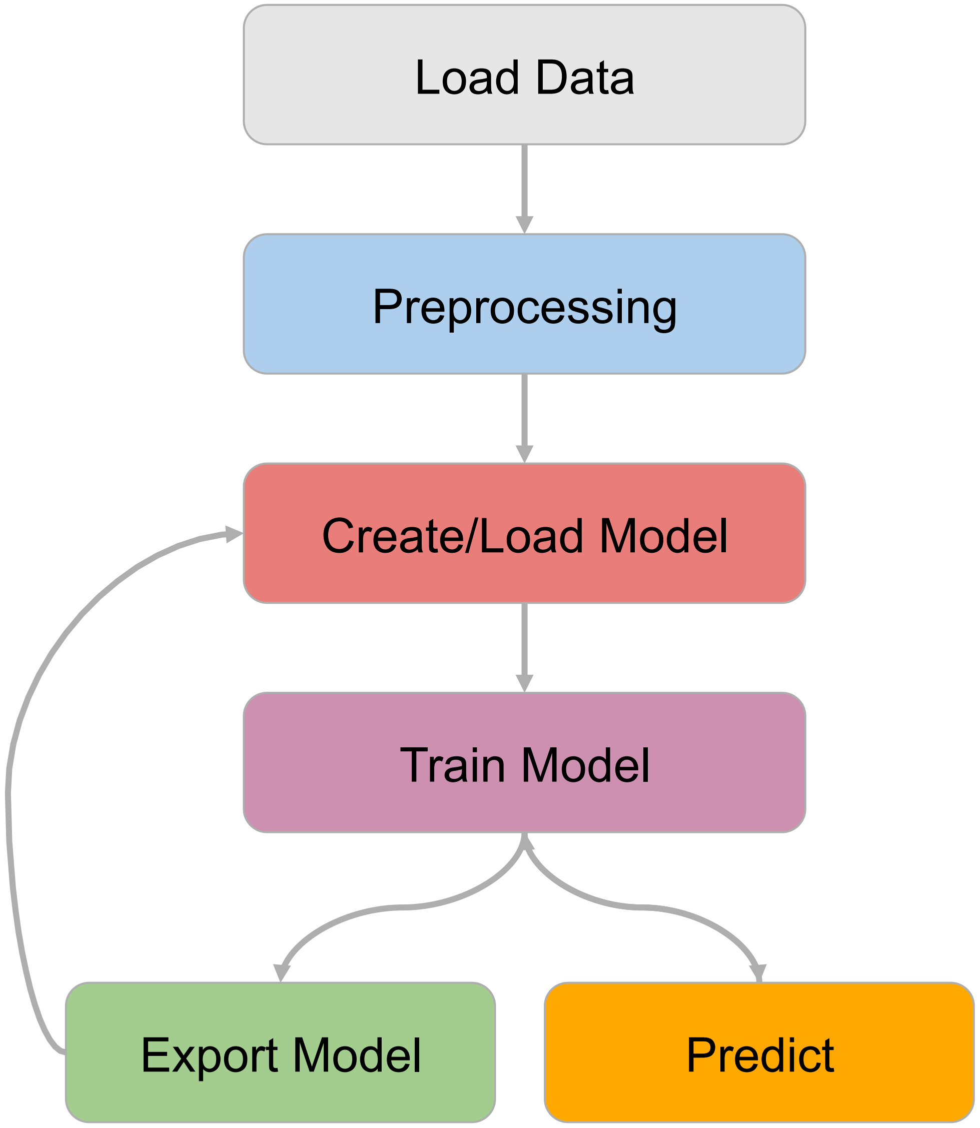
LA libraries | Standard LA APIs | Run-time/ comm. APIs | Vendor Libraries



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

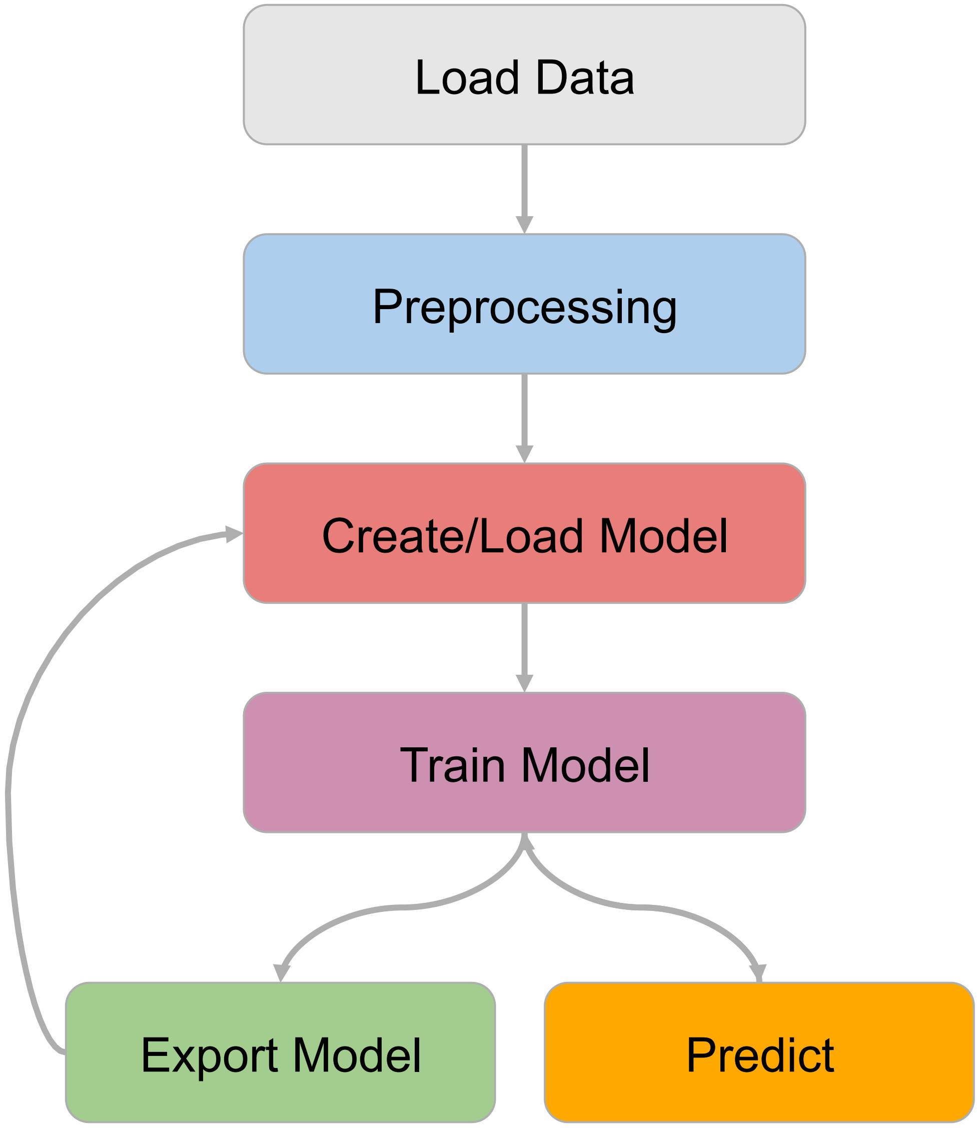
Design process

- Similar to TF or PyTorch
- MagmaDNN is designed/ optimized with this training paradigm in mind. However, it is customizable.

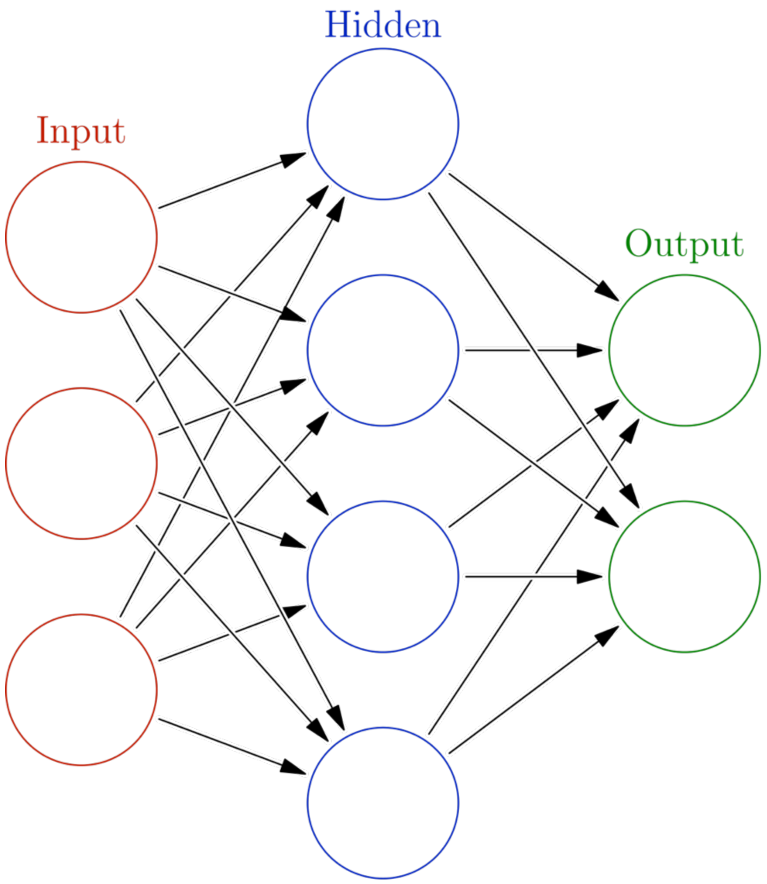


Workflow

- **Load Data**: Read-in any CSV, image, or other file necessary for training.
- **Preprocessing**: Shape data and store in tensors.
- **Create/Load Model**: Restore a saved model or create a new one using MagmaDNN's Model class. Set hyperparameters.
- **Train Model**: Fit the network using SGD.
- **Predict**: Use the fitted weights to predict class based on new input.
- **Export Model**: Save model to be used again.



Neural Network Ideas



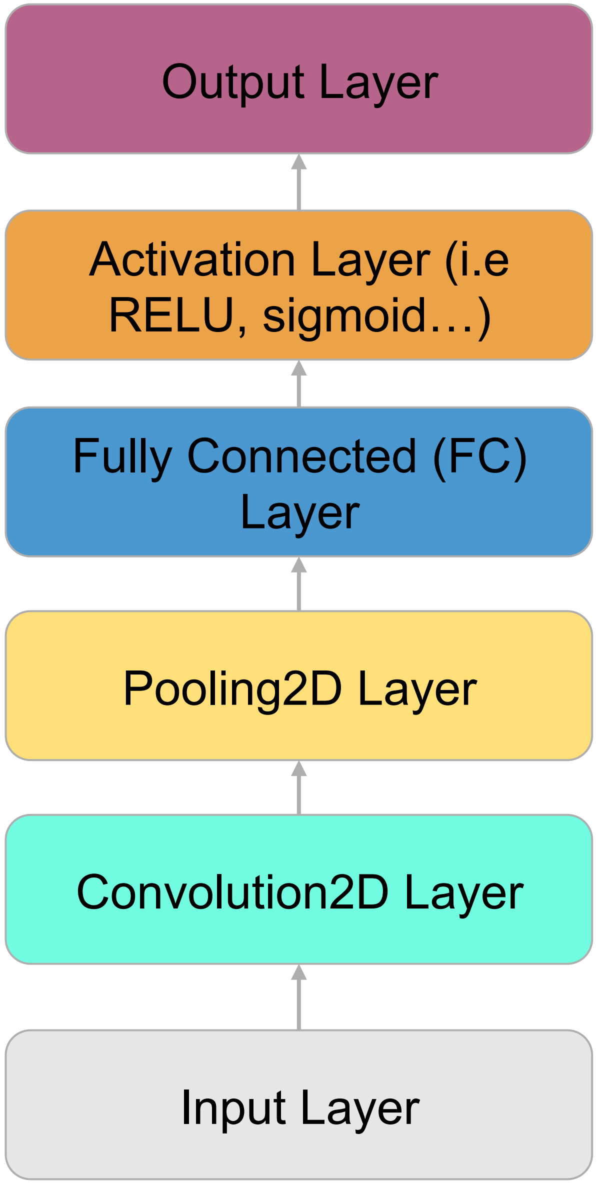
Neural Networks are typically composed of layers of linear transformations wrapped by activation functions. The network is represented by some function *f*.

After optimizing some loss criterion w.r.t. the parameters of *f*, the function (or "network") becomes an accurate predictor of highly abstracted data.

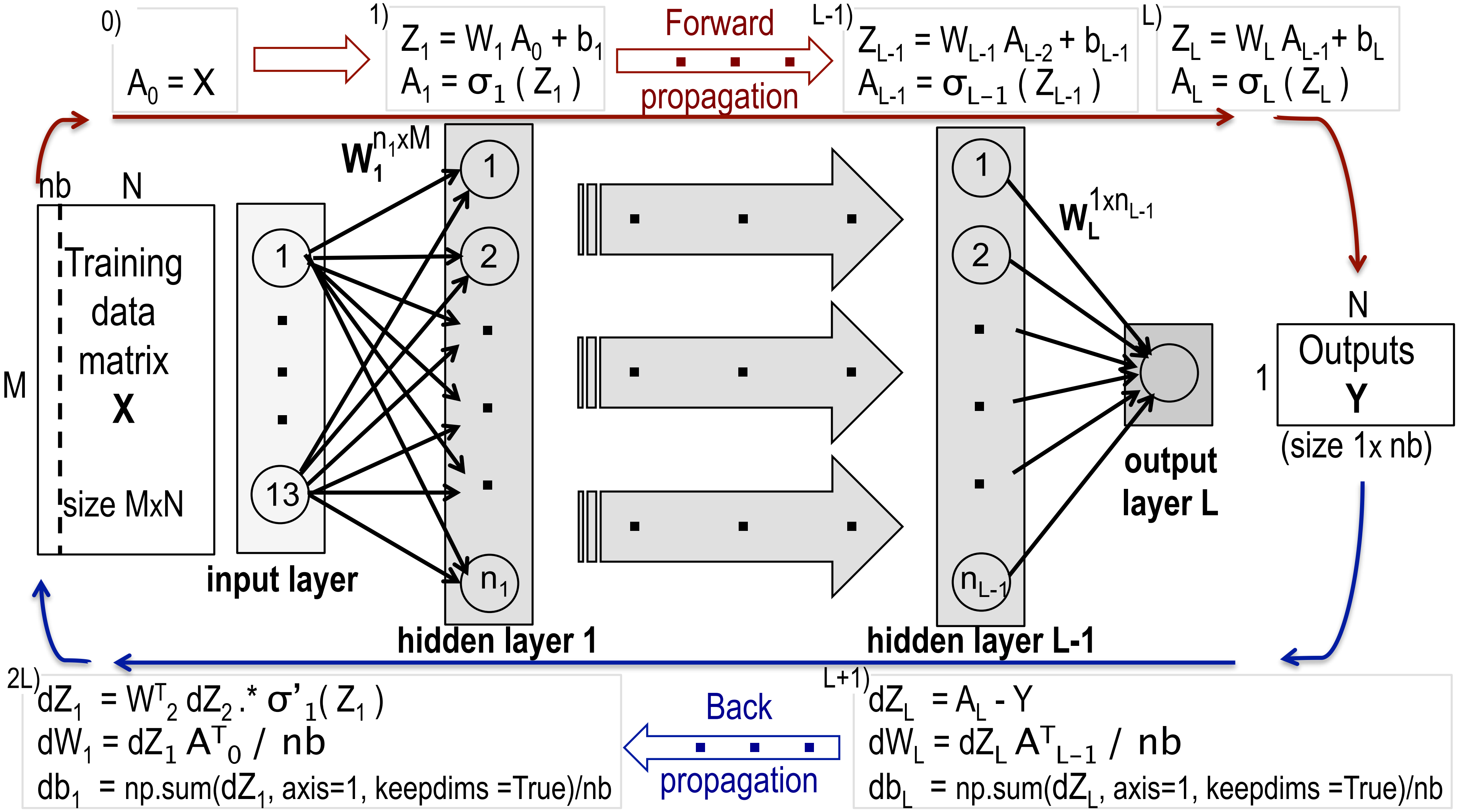
Other common, more complicated network types exist: CNN, RNN, GANs, Belief Networks, Boltzmann

Neural Network Ideas (cont.)

- Layers
  - Neural Networks are comprised of several layers put together.
  - Available Layers:
    - Input, Output (first and last layers of the network)
    - Fully Connected (dense, linear transformation)
    - Activation (activation function)
    - Conv2D, Pooling2D (convolutional layer)

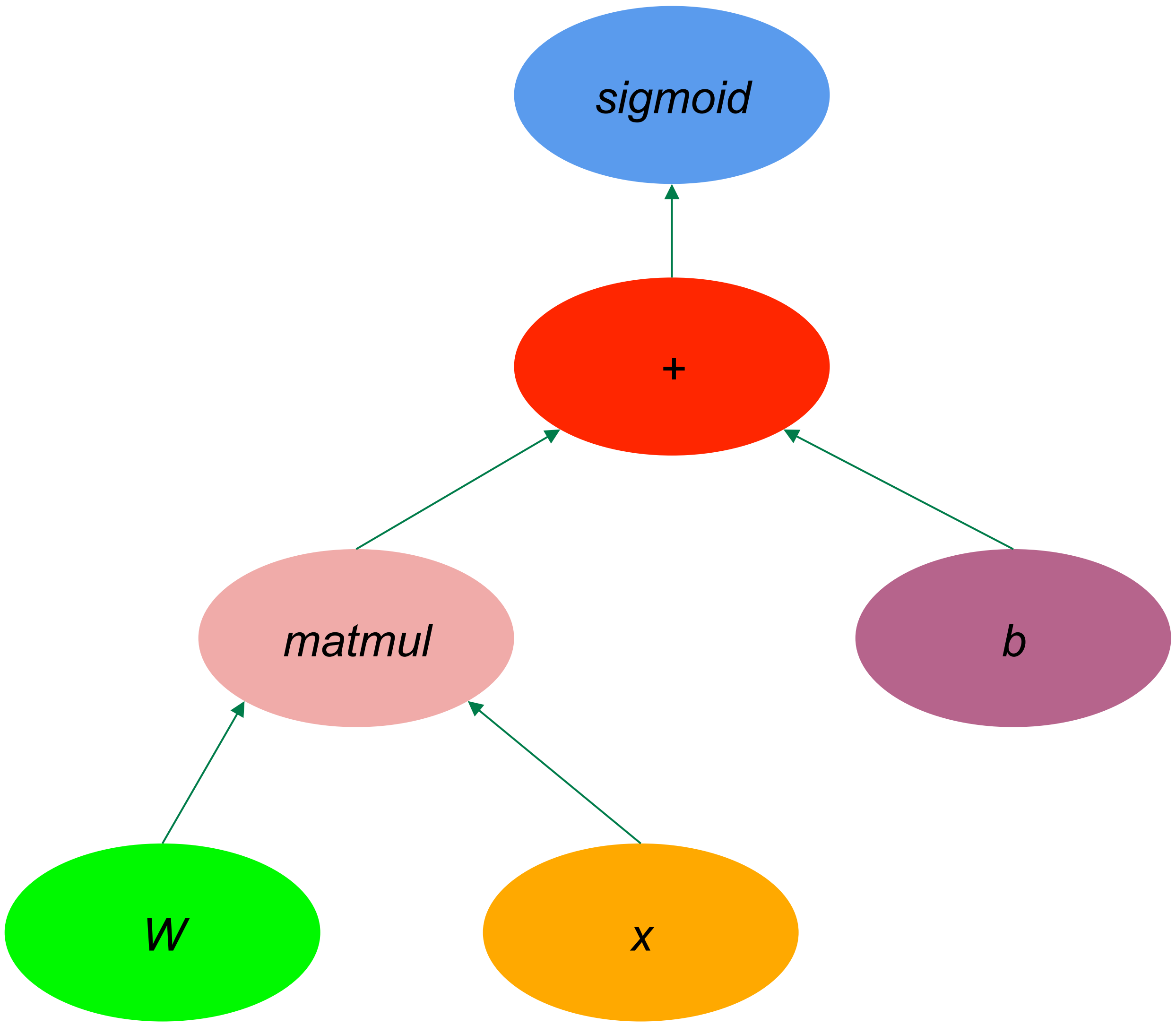


DNN example representation



Compute Graph

- All operations/math are put into a compute graph.
- Non-Eager
- Gradient Support, Grad Tables



Operations & Compute Graphs

All Tensor operations are wrapped in an Operation class, which is used in the compute graph. Operations also provide a modular interface for creating and manipulating Tensors. They are created as shown:

```
Operation<float> *var = op::var<float> ("Var Name", {5, 4}, {GLOROT, {0.5, 0.2}}, HOST);
```

Var creates and returns a new variable

Tensor shape

Tensor initializer. Options are: GLOROT, UNIFORM, CONSTANT, ZERO, ONE, DIAGONAL, IDENTITY, NONE

Tensor memory type. Options are: HOST, DEVICE, MANAGED, CUDA_MANAGED

Operations & Compute Graphs (cont.)

Variables are Operations that wrap around Tensors. Operations are also used for representing some math operation in the computational graph. For example:

```
Operation<float> *result = op::add(op::matmul(A, x), b);
Tensor<float> *result_tensor = result->eval();
```

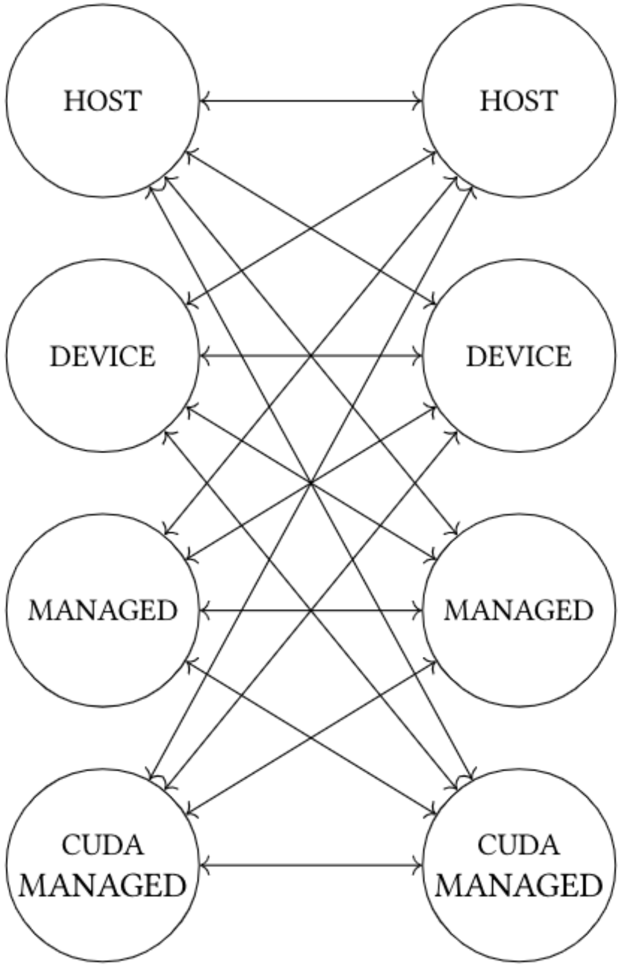
This constructs a compute graph and *eval()* evaluates it into a Tensor. Available operations are: Variable, Tanh, Sigmoid, Add, and Matmul. Since all of these are inherited from Operation, it is simple to create/add new operations.

Operations & Compute Graph (Full Example)

```
auto A = op::var<float> ("A", {4, 5}, {GLOROT, {1.5, 2.0}}, MANAGED);
auto X = op::var<float> ("X", {5, 4}, {UNIFORM, {0.0, 1.0}}, MANAGED);
auto B = op::var<float> ("B", {4, 4}, {DIAGONAL, {1, 2, 3, 4}}, MANAGED);

/* compute some math operations */
auto result = op::add(op::matmul(A, X), B);
Tensor<float> *result_tensor = result->eval();

/* use results .... */
delete result;  /* only need to delete head of tree */
delete result_tensor;
```

Memory Manager

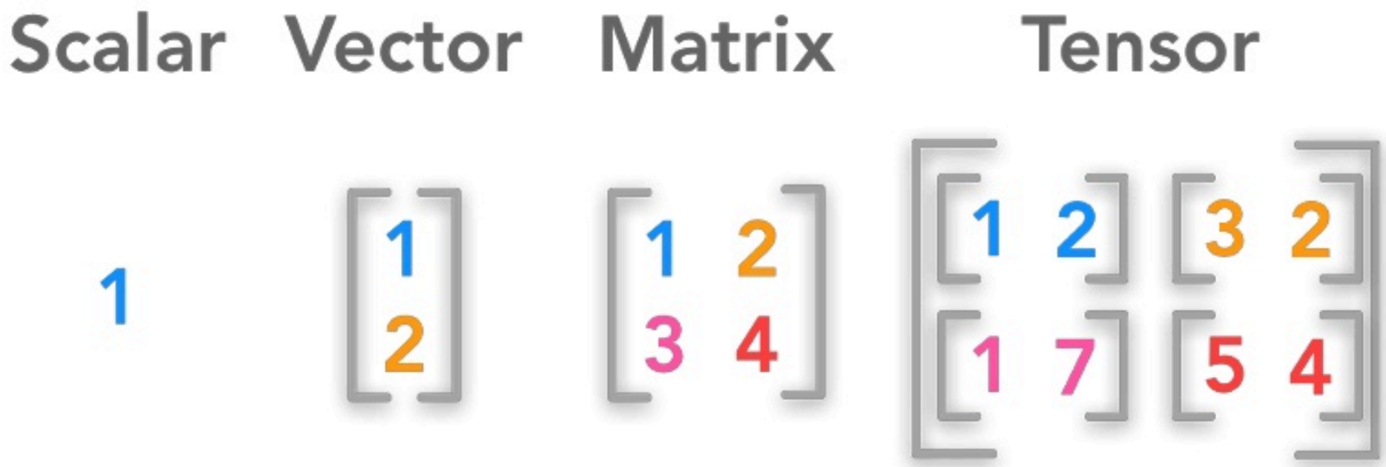
- Core Memory Kernel
- 4 memory types:
  - HOST (cpu memory)
  - DEVICE (gpu memory)
  - MANAGED (internal managed)
  - CUDA_MANAGED (cuda managed)
- Supports interactions between all memory types
- Managed memory types must be synced!



Tensors

Data with multiple axes.

Everything in MagmaDNN uses tensors.



Layers

Layers are a set of weights/biases and put a forward-prop function on the compute graph. For instance:

```
layer::FullyConnectedLayer<float> *fc = layer::fullyconnected(input->out(), n_units);
```

This creates a weight, w, and bias, b, tensor and puts [*W*input->out()* + *b*] onto the head of the compute graph defined by *input->out()*.

Layers (Full Example)

```
auto data = op::var<float> ("data", {n_batches, size}, {UNIFORM, {-1.0, 1.0}}, DEVICE);

auto input = layer::input(data);
auto fc1 = layer::fullyconnected(input->out(), n_hidden_units);
auto act1 = layer::activation(fc1->out(), layer::TANH);
auto fc2 = layer::fullyconnected(act1->out(), n_output_classes);
auto act2 = layer::activation(fc2->out(), layer::SIGMOID);
auto output = layer::output(fc2->out());

Tensor<float> *forward_prop_result = output->out()->eval();
```

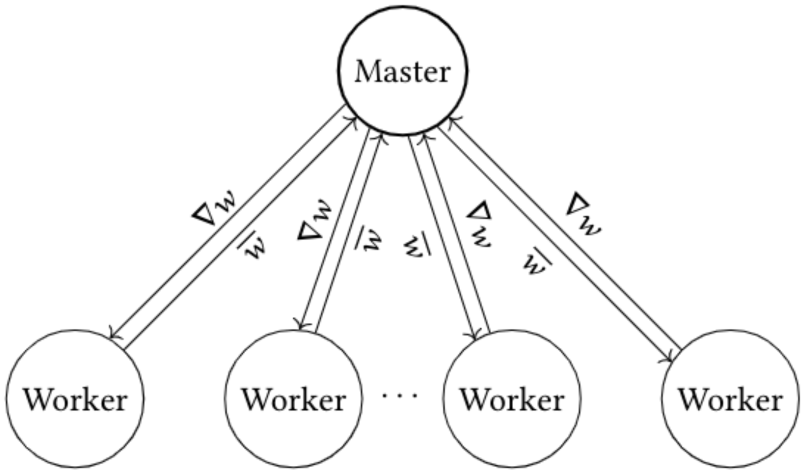
Training (example)

```
Tensor<float> data  ({60000, 785}, HOST);
io::read_csv_to_tensor(data, "mnist_data_set.csv");

std::vector<Layer<float>> layers_vector;
/* Create Layers in Here as Shown Before... */

Optimizer<float> optimizer = optimizer::DistributedGradientDescentOptimizer(0.05);
Model<float> model (layers_vector, optimizer, batch_size);
model.fit(data, n_epochs);
```

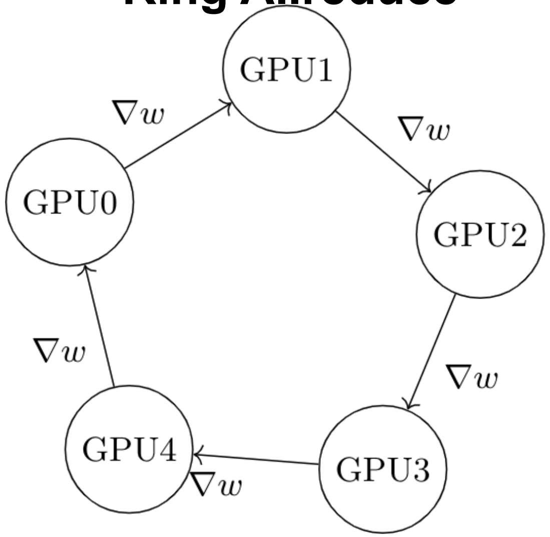
Distributed Training

- Many node training
- Averages gradients
- Implemented many strategies and optimizations (using CUDA-aware MPI)

Master-worker reduce



Ring Allreduce



MPI_Allreduce

Asynchronous training

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**

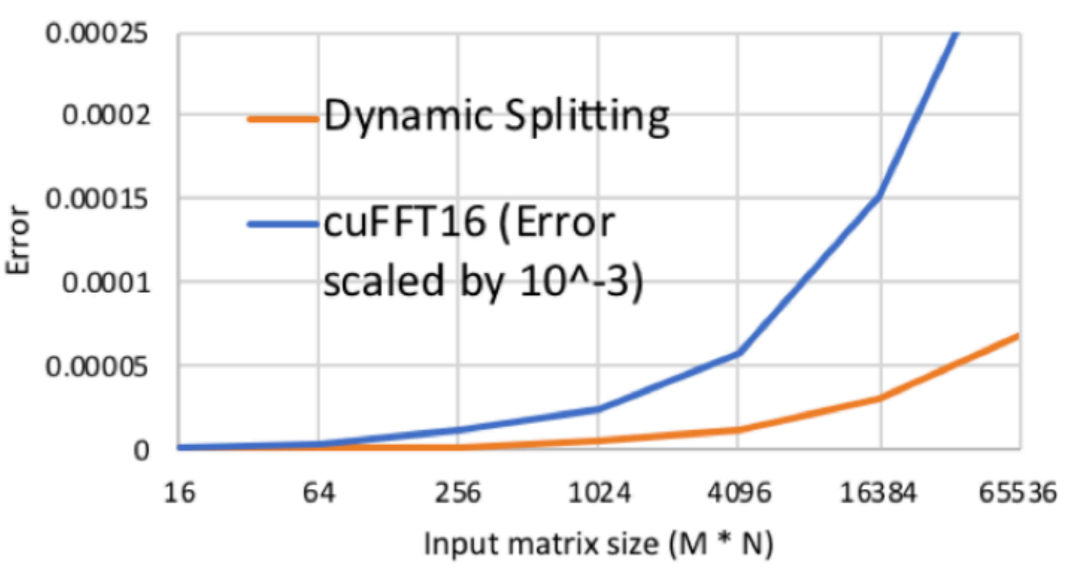
$$X_{FP32}(:) = s_1\, X1_{FP16}(:) + s_2\, X2_{FP16}(:)$$

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

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

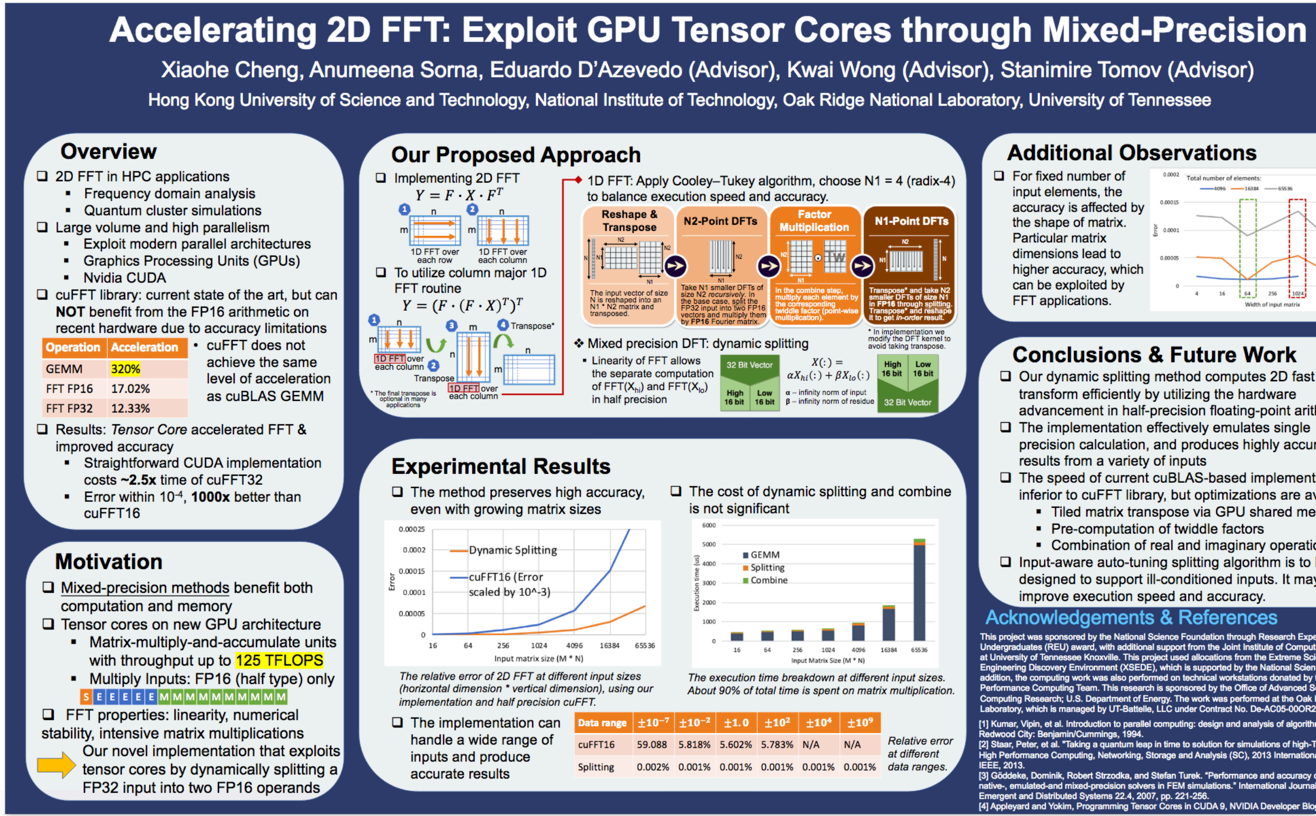
Accelerating CNNs with FFT

➢ **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

  $$\mathbf{D * F = A^T \, [\, [\, G \, D \, G^T] \, .* \, [B^T \, D \, B] \,] \, A}$$

  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**	
…				

Install and Build

Dependencies:

- Cuda (>9.0)
- CuDNN (>6.0)
- Magma (>2.3.0) (>2.5.0 for half-precision)

Download MagmaDNN from https://bitbucket.org/icl/magmadnn (currently not up to date) or clone it using

hg clone https://bitbucket.org/icl/magmadnn

Compiling/Installing: Copy the *make.inc* file from *make.inc-examples/* to MDNN's root, change any necessary settings in *make.inc* and then run

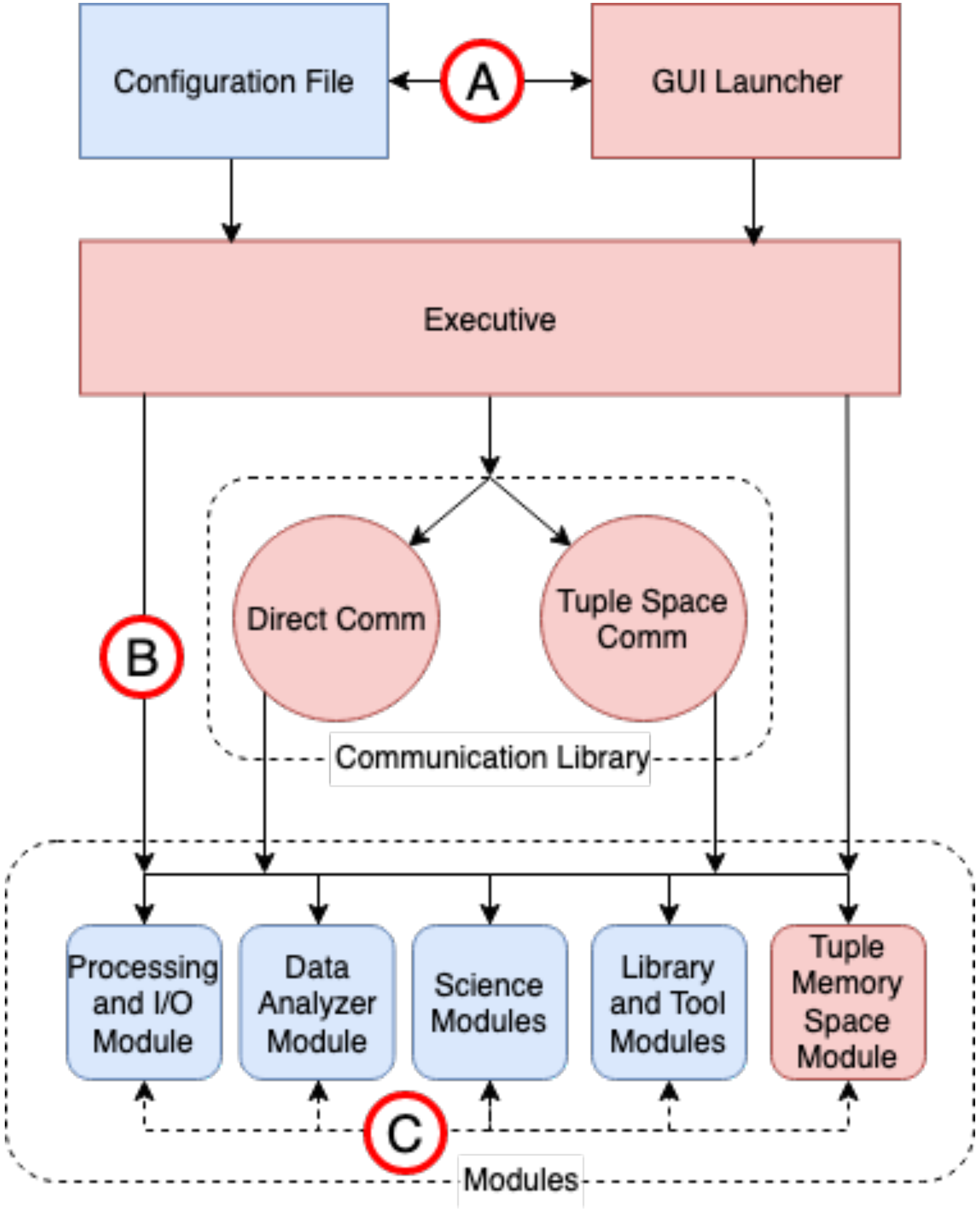
sudo make install

Testing: You should now be able to run the below command

make testing && cd testing && sh run_tests.sh

this will run the default testers for the MagmaDNN package.

Hyperparameter optimization



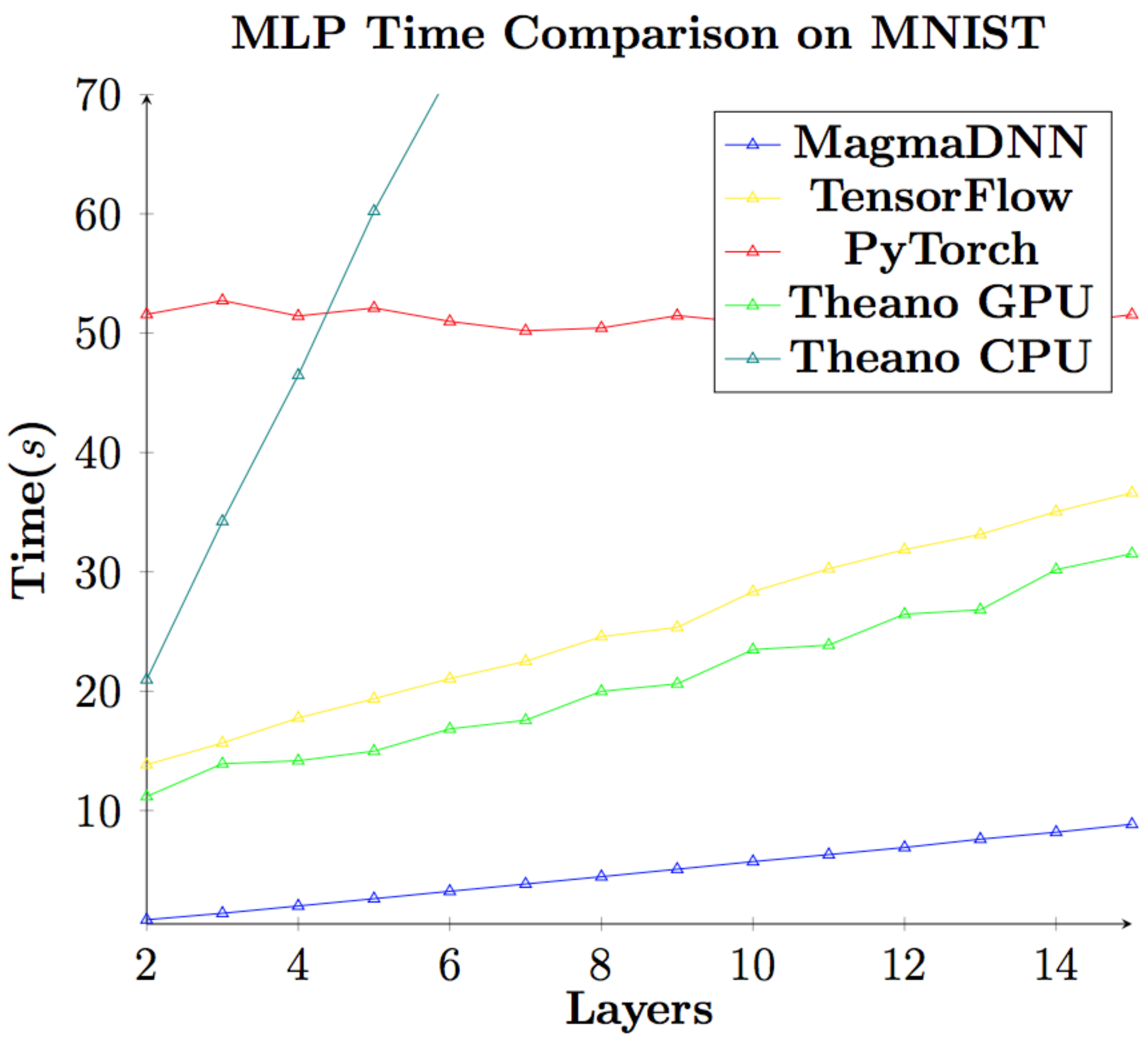
OpenDIEL architecture:

(A) GUI launcher creates a configuration file for the workflow, and executive will read this file to set up workflows;

(B) After initial configuration, executive starts all modules;

(C) The modules have access to the communication library, and directly communicate or utilize tuple-space communication.

MagmaDNN training performance (single V100 GPU)



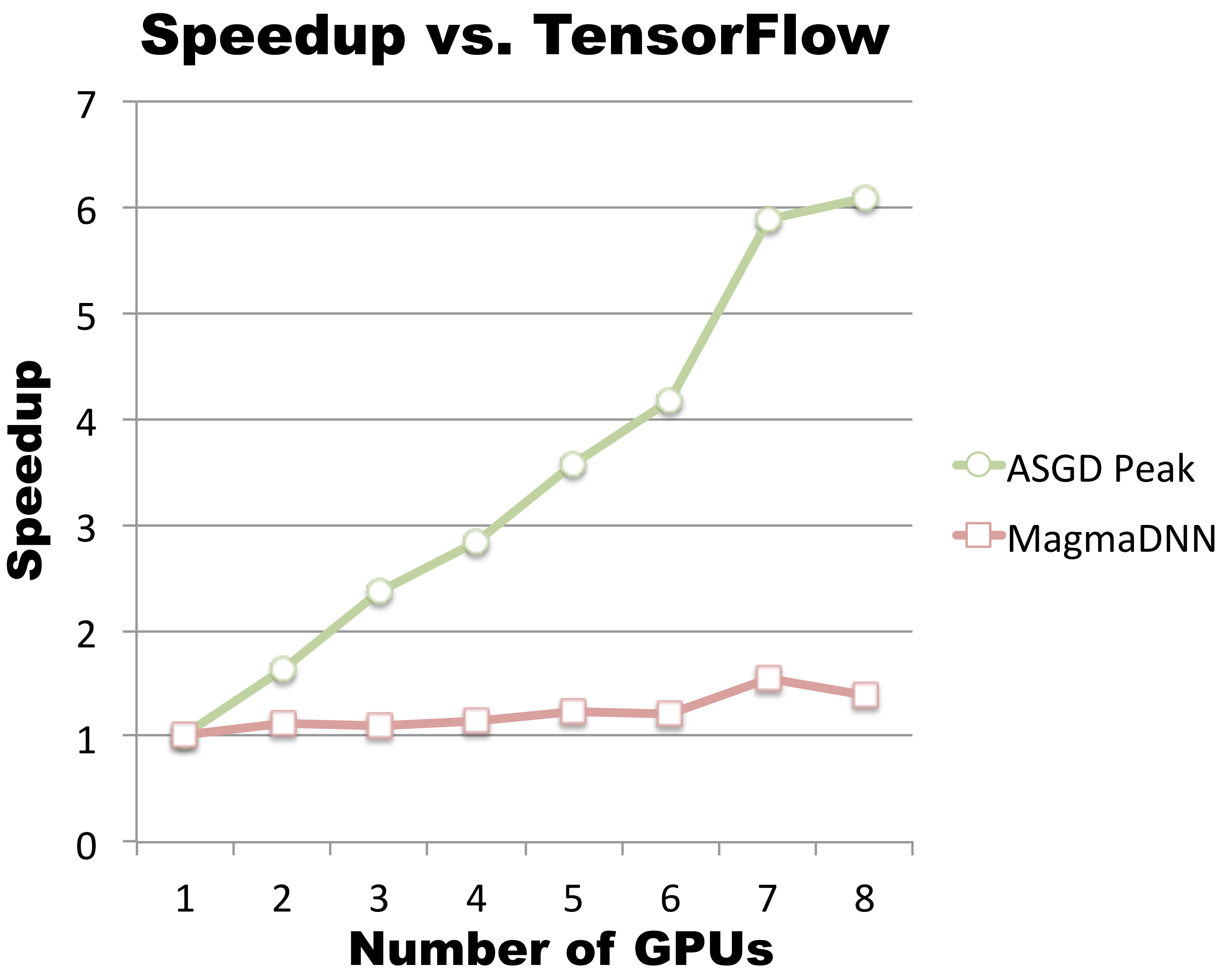
Data: 60,000 images, 28x28 pixels each

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
$\frac{\#\text{Hidden Units}}{\text{Layer}}$	528

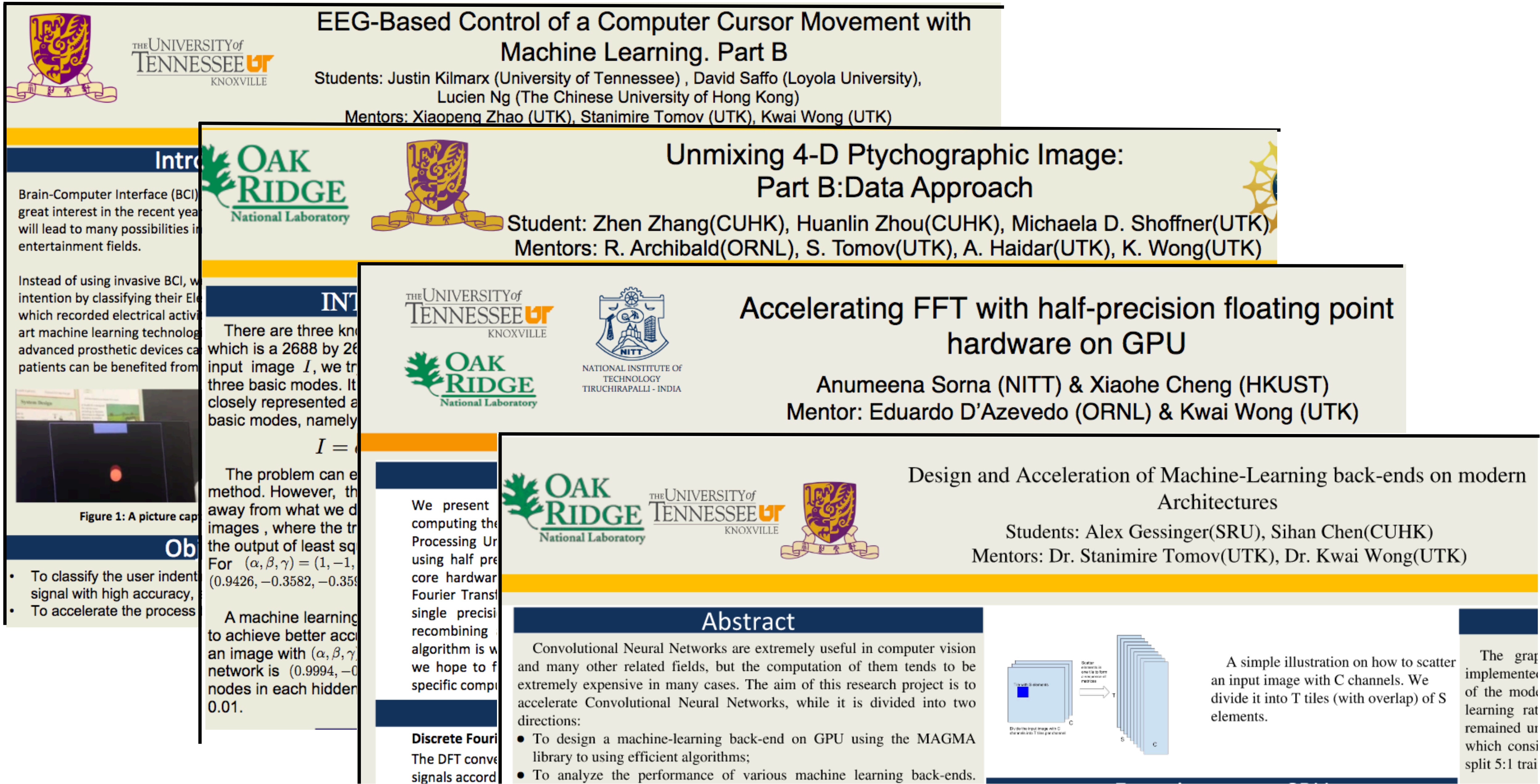
ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE Department of Electrical Engineering and Computer Science

MagmaDNN scalability and SGD speedup



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.
  - Shows potential; still lacks the arsenal of features present in other popular frameworks
- **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
  - Distribution strategies and (asynchronous) techniques for training DNN on large scale systems
- **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
LLNL
ORNL
ANL
SANDIA
University of California, Berkeley
University of Colorado, Denver
TAMU
INRIA, France
KAUST, Saudi Arabia
University of Manchester, UK

CEED: Center for
Efficient Exascale Discretizations