

Integrating Deep Learning in Domain Science at Exascale (MagmaDNN)

Stan Tomov

Research Assistant Professor
University of Tennessee

Rick Archibald, Eduardo D'Azevedo, Markus Eisenbach, and Junqi Yin
Oak Ridge National Laboratory

Edmond Chow, Georgia Institute of Technology

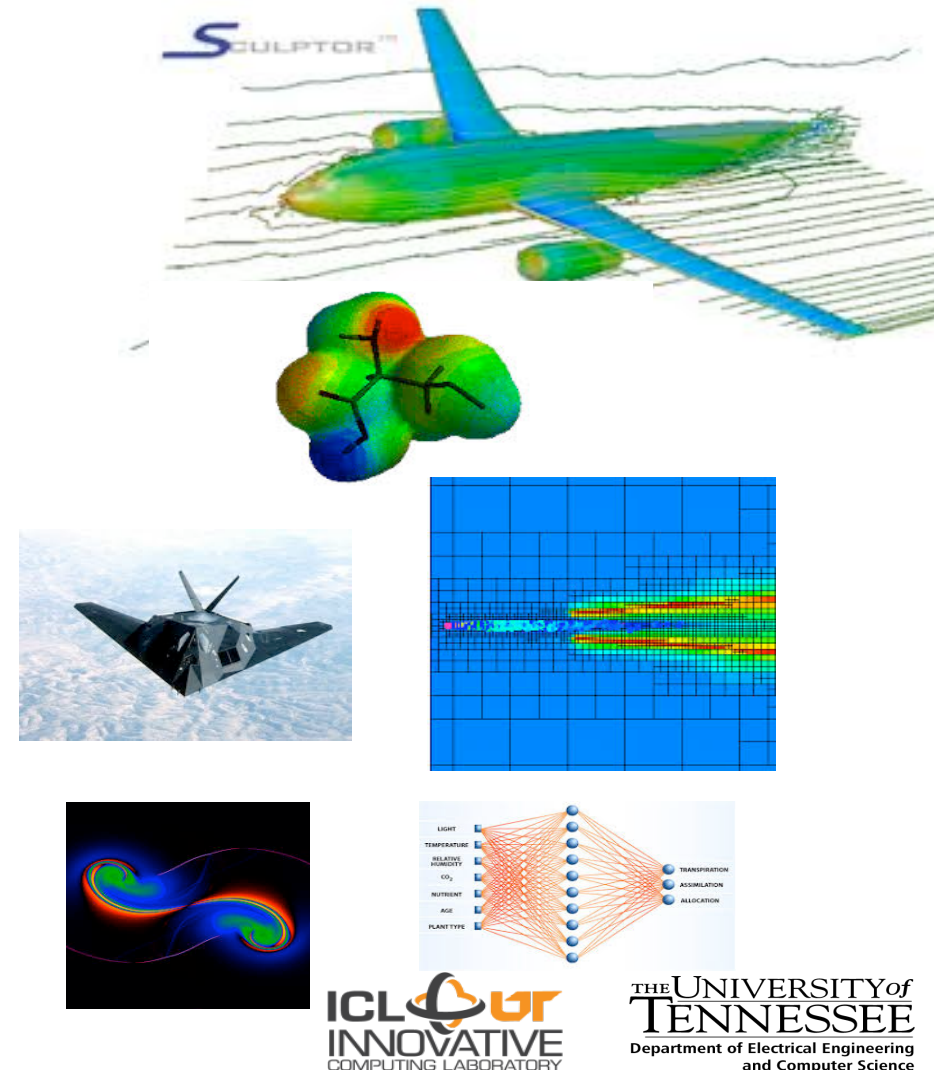
Jack Dongarra, Rocco Febbo, Florent Lopez, Daniel Nichols, and Kwai Wong
University of Tennessee, Knoxville

DOD HPCMP virtual seminar
December 8, 2020

Dense Linear Algebra

Needed in a wide variety of domain sciences
Can power ML and data analytics too:

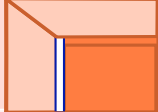

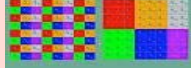

- **Linear systems:** **Solve $Ax = b$**
 - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** **Find x to minimize $\|Ax - b\|$**
 - Computational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



High-performance LA for modern architectures

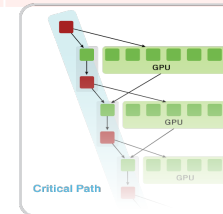
- Leverage latest numerical algorithms and building blocks
MAGMA, PLASMA, SLATE (DOE funded),
MAGMA Sparse, POMPEI project*
- Polymorphic approach
Use MAGMA sub-packages for various architectures;
Provide portability through single templated sources using C++
- Programming model
BLAS tasking + scheduling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

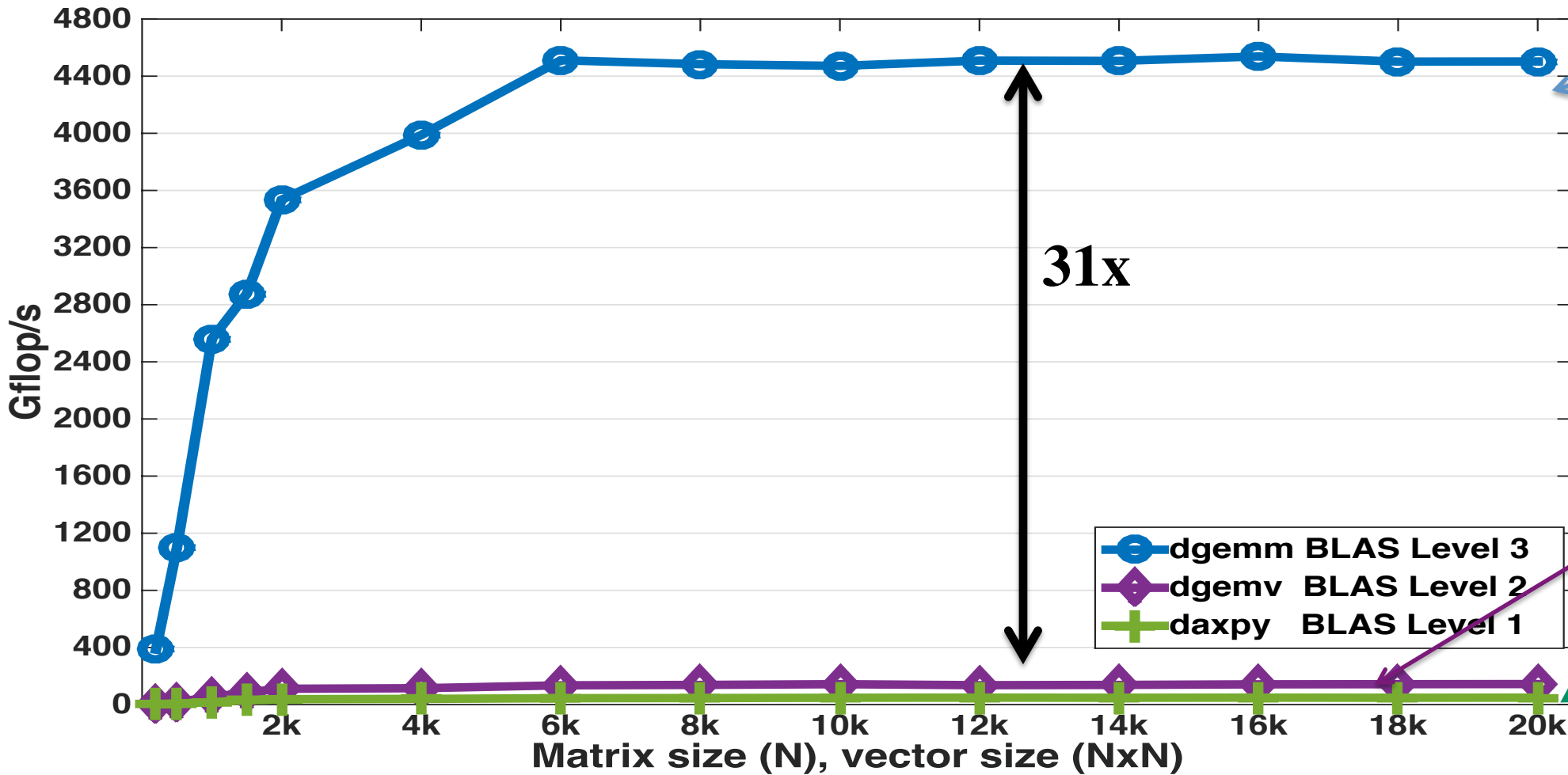
Use of BLAS is in the heart of ML performance and portability too!

HPC software design – use Level 3 BLAS

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



$C = C + A * B$
4503 Gflop/s



$y = y + A * x$
145 Gflop/s

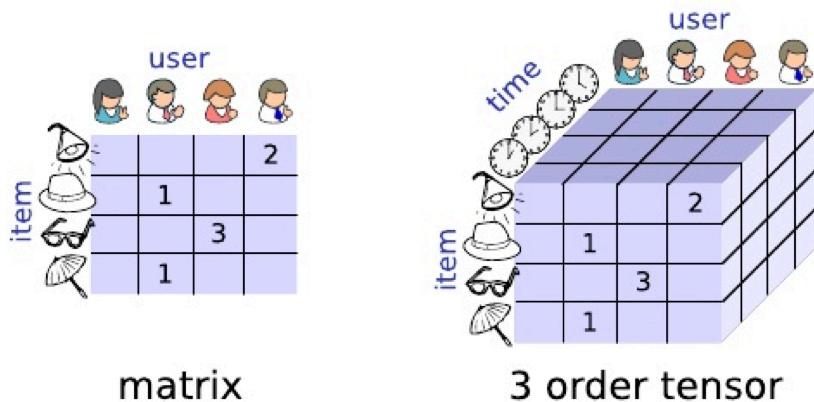
$y = \alpha * x + y$
52 Gflop/s

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

What other LA is needed 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 (Batched LA)

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

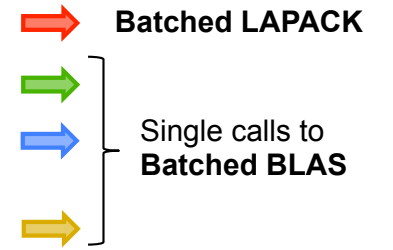
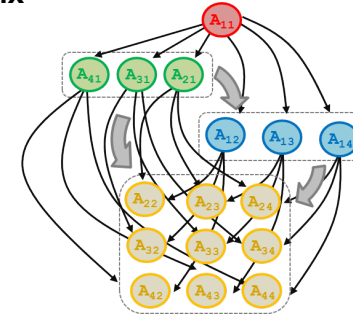
- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc.

Sparse/Dense solvers & preconditioners

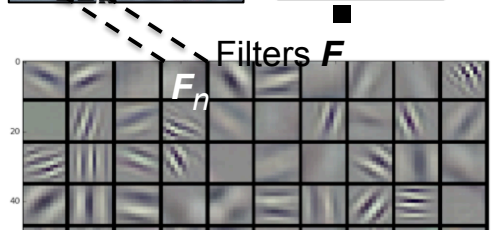
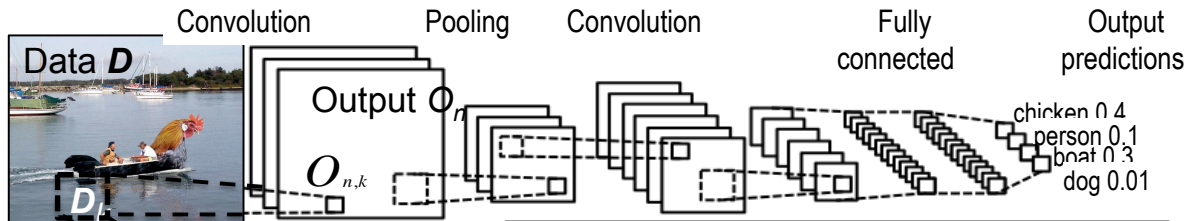
Sparse / Dense Matrix System

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

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:

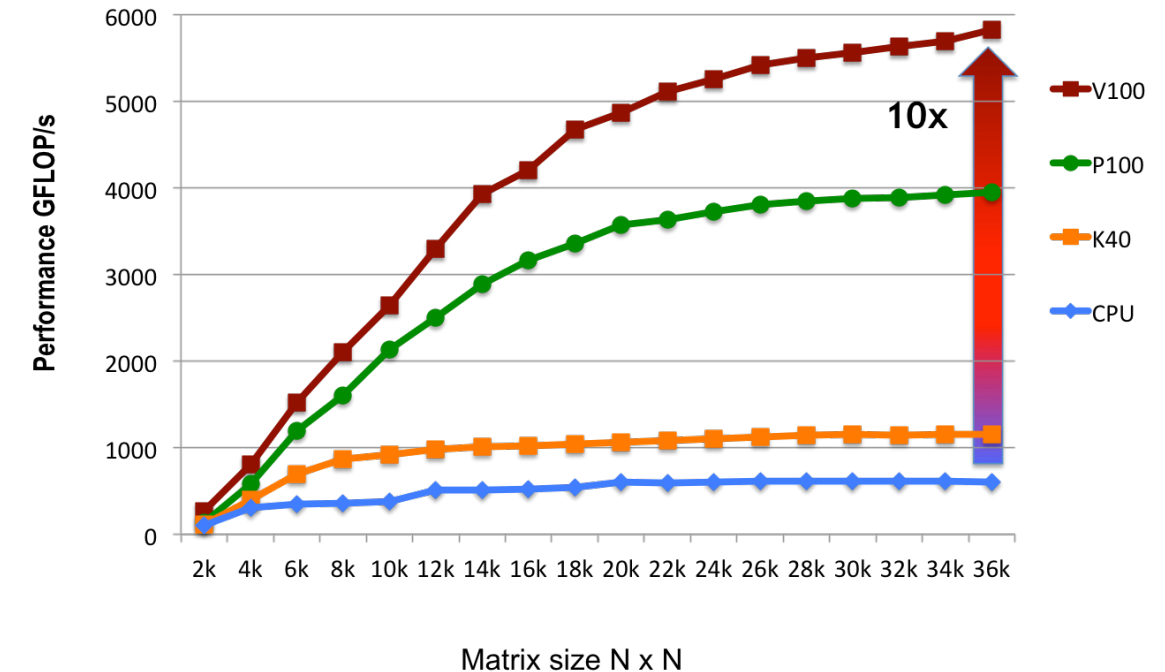
$$\text{Batch}_{\{ C_{i3} = A^T B_{i3}, \text{ for range of } i3 \}}$$

Main Classes of Algorithms in MAGMA

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

MAGMA 2.3 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

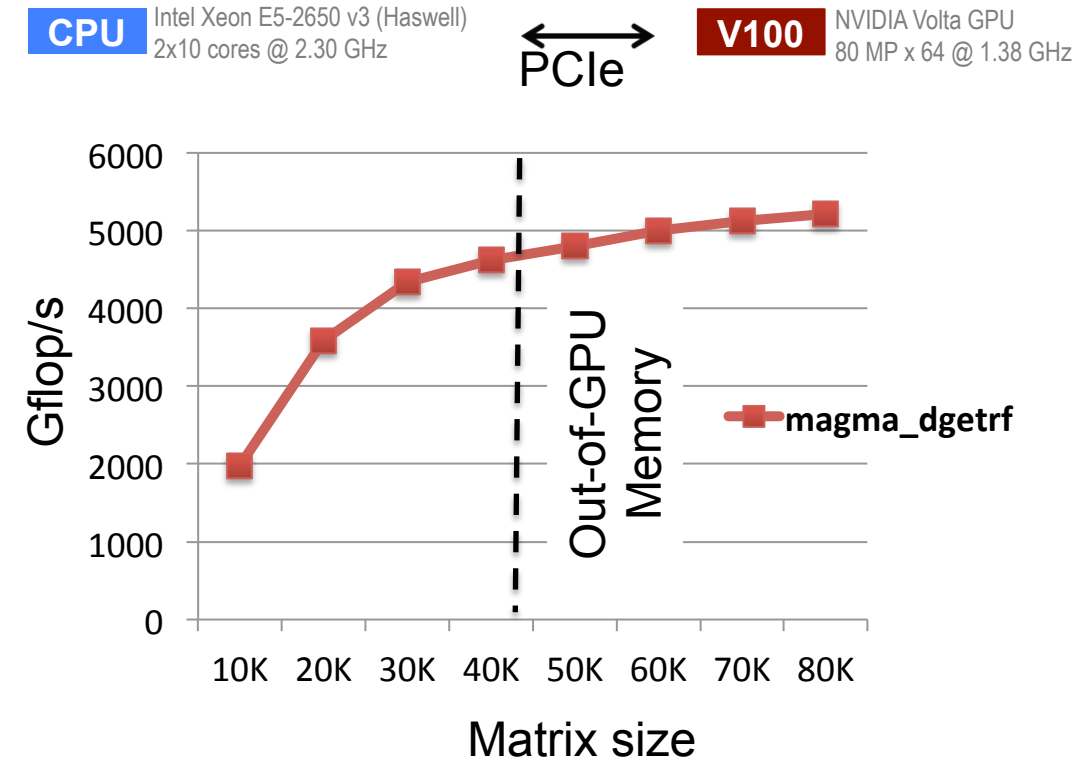


Main Classes of Algorithms in MAGMA

- 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.

Performance of LU in DP

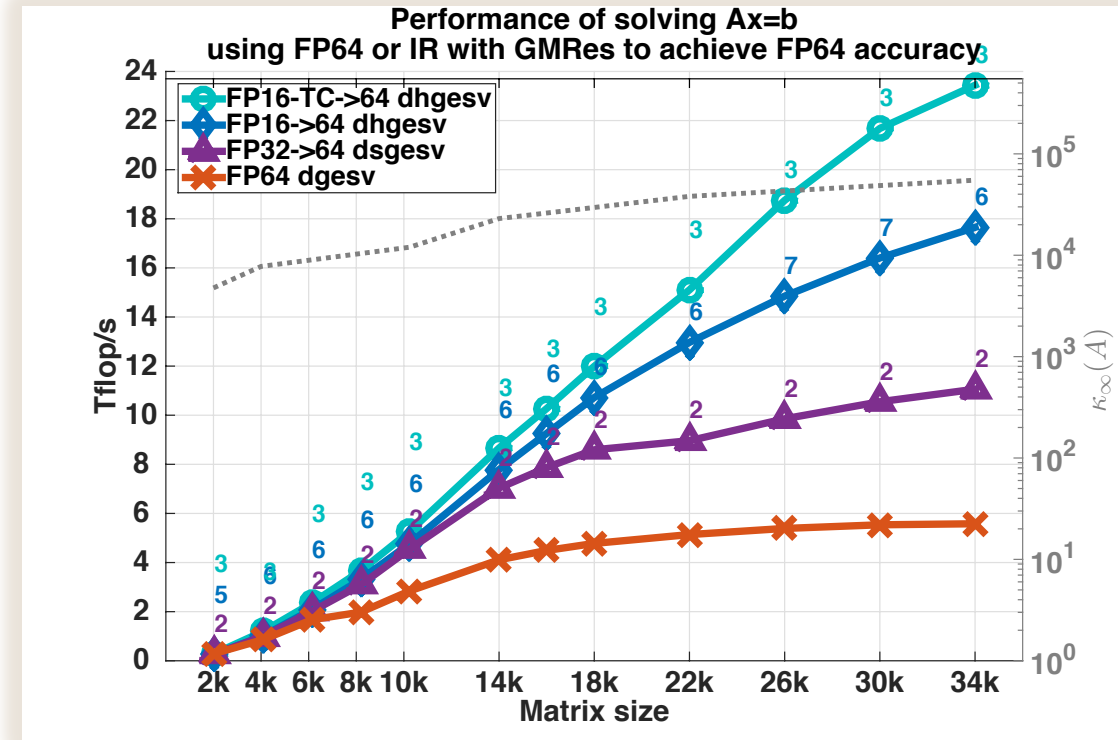


Main Classes of Algorithms in MAGMA

- 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, “Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers”, SC’18 (accepted), November 2018.

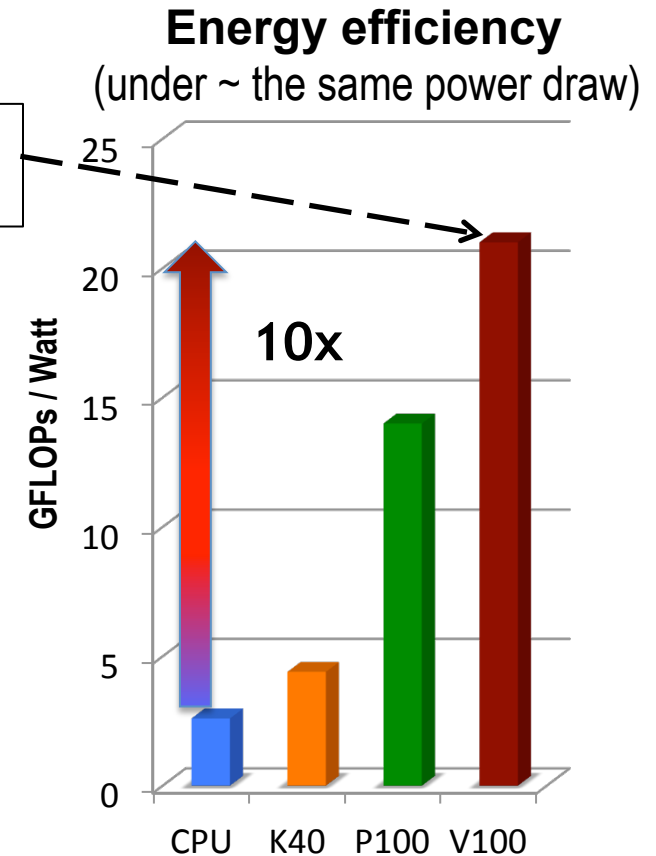
Posters (GTC’18 2nd place, ISC’18 1st place; 11K downloads in a month)



Main Classes of Algorithms in MAGMA

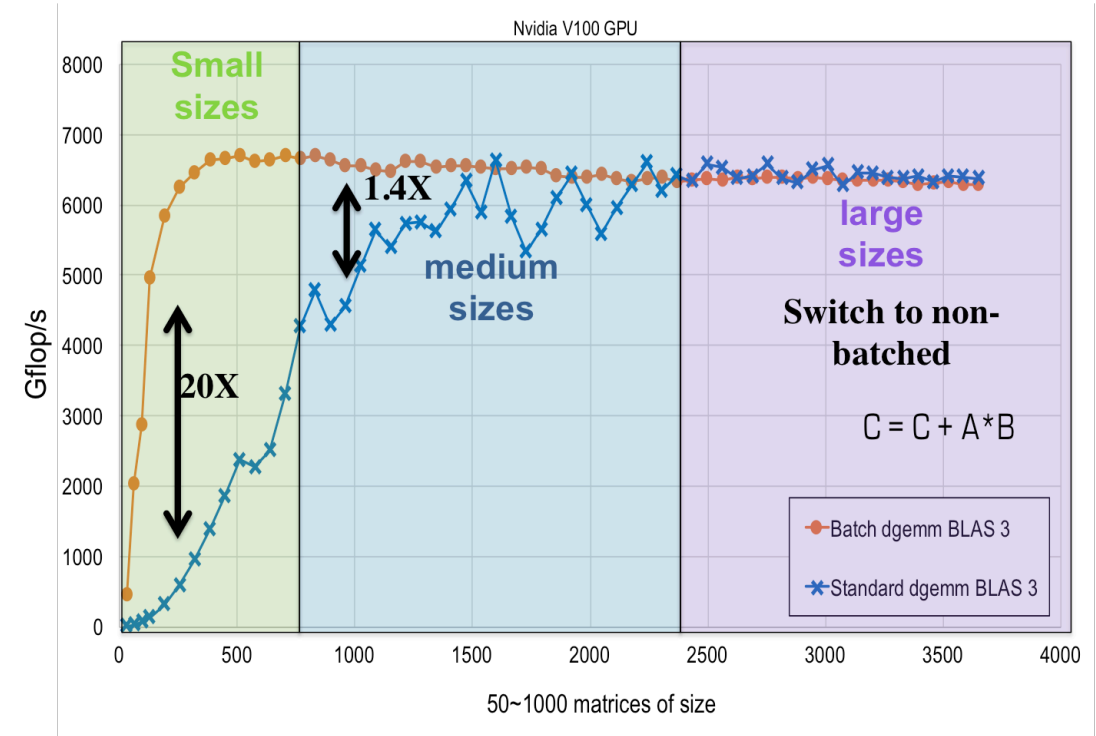
- Hybrid algorithms
 - Use both CPUs and GPUs
- GPU-only algorithms
 - Entirely GPU code
- Out-of-GPU memory algorithms
 - LA that is too large to fit into the main CPU/GPU memory
- Mixed-precision LA
 - Use new hardware features, e.g., Tensor Cores
- Energy efficient
 - Build energy awareness and tradeoff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

... and **76 Gflop/Watt** using mixed-precision !



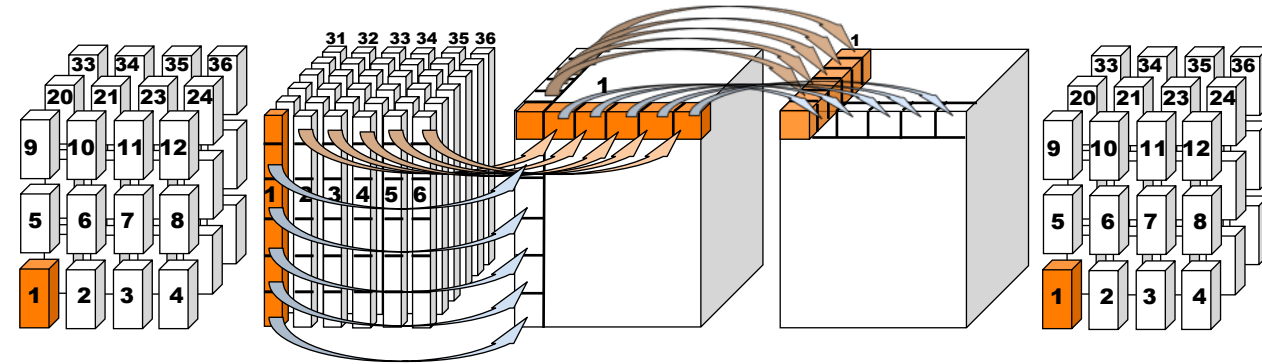
Main Classes of Algorithms in MAGMA

- 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

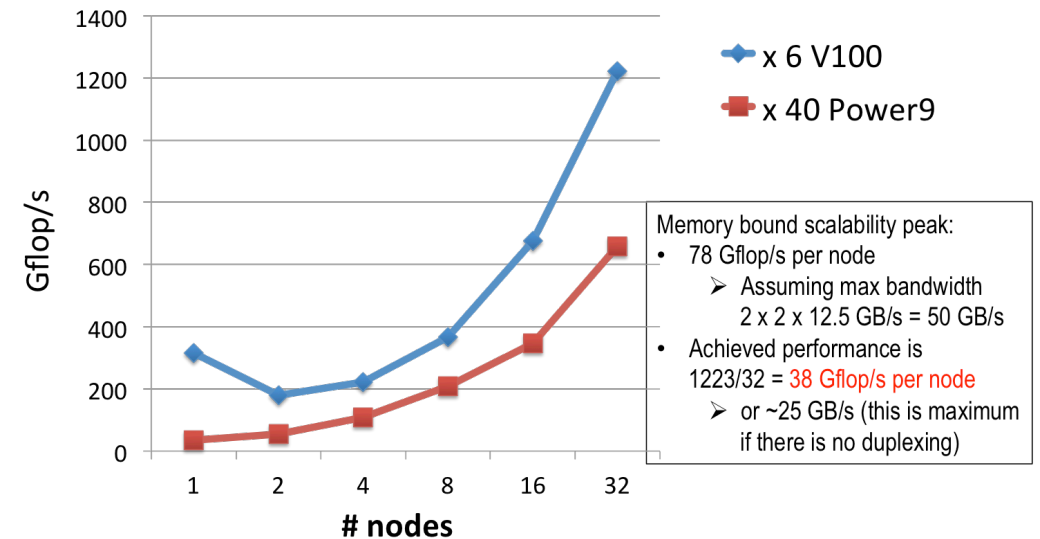


Main Classes of Algorithms in MAGMA

- 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.)

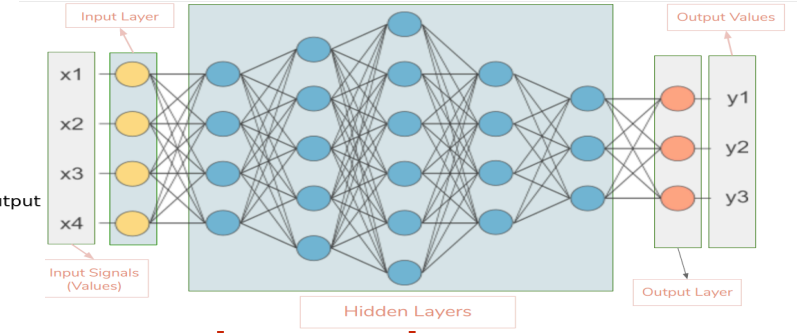
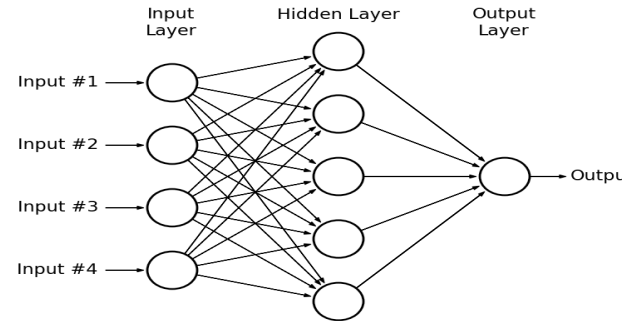
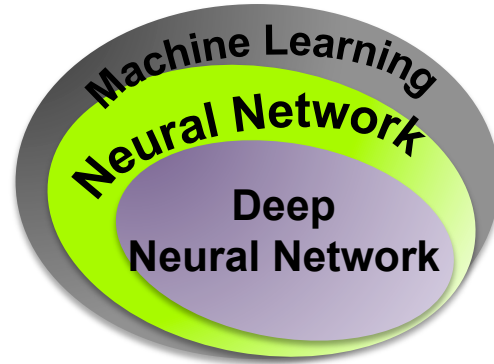
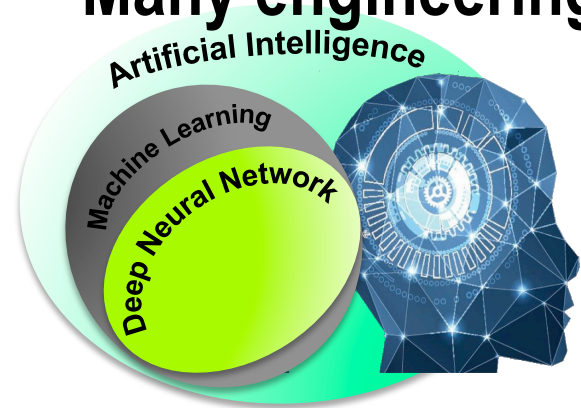


Strong scalability of 3D FFT on Summit (N = 1024)



AI, ML, NN, DNN, data analytics

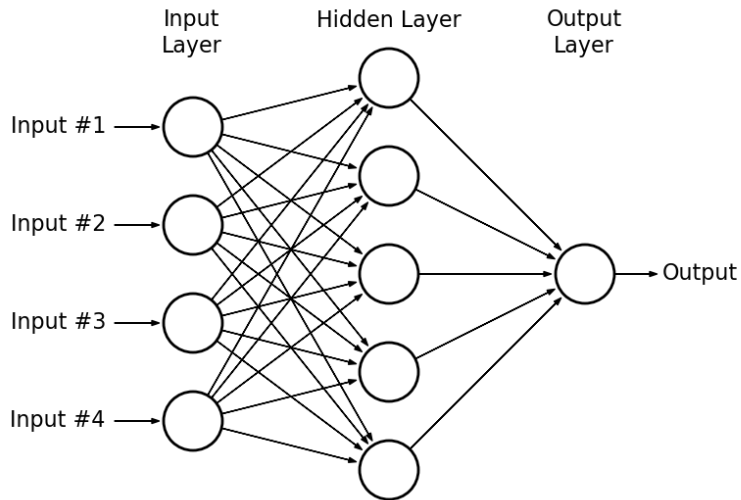
Many engineering problems can be solved using data-driven AI-based simulations



- ✓ **Artificial Intelligence (AI)**: science and engineering of making intelligent machines to perform the human tasks (John McCarthy,1956). AI applications is ubiquitous.
- ✓ **Machine learning (ML)** : A field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959). A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E (Tom Mitchell,1998).
- ✓ **Neural Network (NN)** : Neural Network modeling, a subfield of **ML** is algorithm inspired by structure and functions of biological neural nets
- ✓ **Deep Neural Network (DNN)** : (aka deep learning): an extension of **NN** composed of many layers of functional neurons, is dominating the science of modern **AI** applications
- ✓ **Supervised Learning (SL)** : A class in ML, dataset has labeled values, use to predict output values associated with new input values.

How to build a NN

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - f(w, b, x)\|^2$$



- ✓ Define a **cost/loss function C** , e.g., the *mean squared error (MSE)*.
- ✓ **Minimize $C(w, b)$** as a function of the weights (**w**) and biases (**b**), casting it as an optimization problem using the **gradient descent algorithm**.

- ✓ Define a NN to compute predictions **$f(w, b, x)$** as function of w , b , and data x
- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Weights (**$w = w - \lambda \nabla C$**) and bias (**b**) are the sets of parameters to be determined
- ✓ Many nodes form a **neural layer**, **links** connect layers together, defining a NN model
- ✓ Activation function (**σ**), is generally a nonlinear data operator which facilitates identification of complex features.

- ✓ To compute the gradient ∇C we need to compute the gradients ∇C_x separately for each training input, x , and then average them, $\nabla C = 1/n \sum \nabla C_x$. Unfortunately, when the number of training inputs is very large this can take a long time, and learning thus occurs slowly.
- ✓ A way is to use **stochastic gradient descent** to speed up learning. The idea is to estimate the gradient ∇C by computing a small sample of randomly chosen training inputs, refer to as a **mini-batch** of input (mini-batched SGD).
- ✓ By averaging over this small sample it turns out that we can quickly get a good estimate of the true gradient ∇C , and this helps speed up gradient descent, and thus learning.
- ✓ Another randomly chosen mini-batch are selected and trained, until all the exhausted the training inputs are used. It is said to complete an **epoch (iteration)** of training, then more iteration.

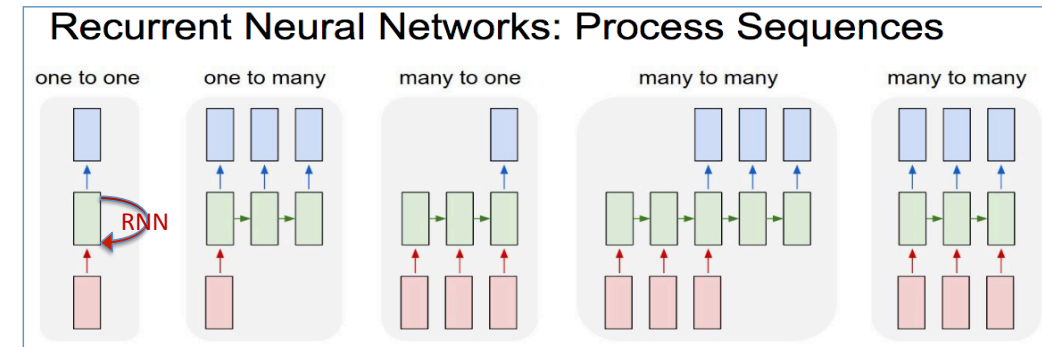
How to build a NN

- There are many parameters determining a NN and possible applications
 - How many layers, what type of layers, sizes, connectivity, what computation graph, activations functions, etc. (model hyperparameters)
 - Algorithm hyperparameters – related to training, like learning rate, mini-batch, etc.

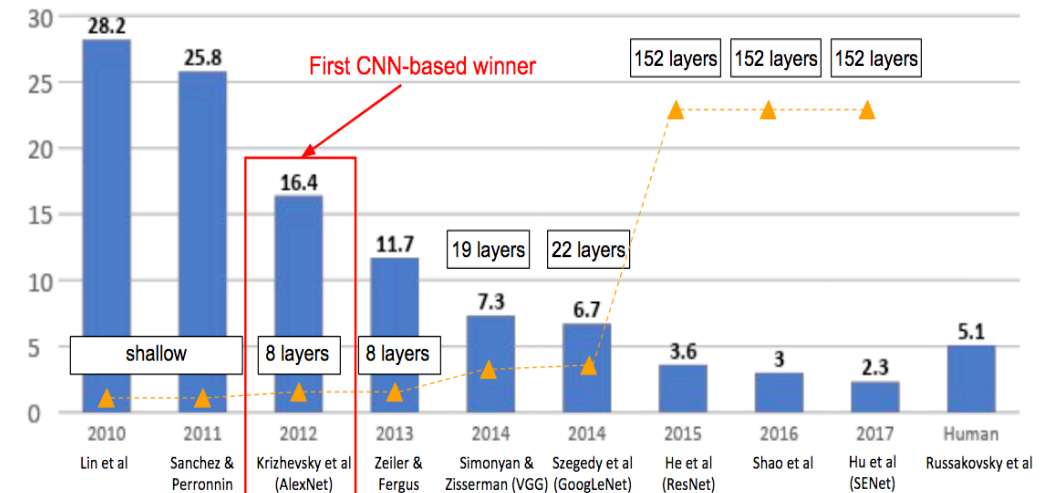
- Examples of different NNs and applications

- **Multilayer Perceptron (MLP) and Convolutional NNs (CNNs)**
have "regular (steady) input and output" (feedforward/no cycles)
Mainly used for regression and image classification
- **Recurrent Neural Networks (RNNs)**
have "time (step) dependent varying size of input" and "irregular" output
Used in speech, text, image, video recognition/classification
- **Generative Adversarial Networks (GANs)**
have a pair of NNs gaming against each other – learning to generate new data with the same statistics as the training set
Used in unsupervised learning, semi-supervised learning, fully supervised learning, and reinforcement learning.
- **Reinforcement Learning**
Software agents take actions in order to maximize the notion of cumulative reward
Used in robot control, elevator scheduling, telecommunications, backgammon, checkers and Go
- ...

- Networks become more complicated, DNNs

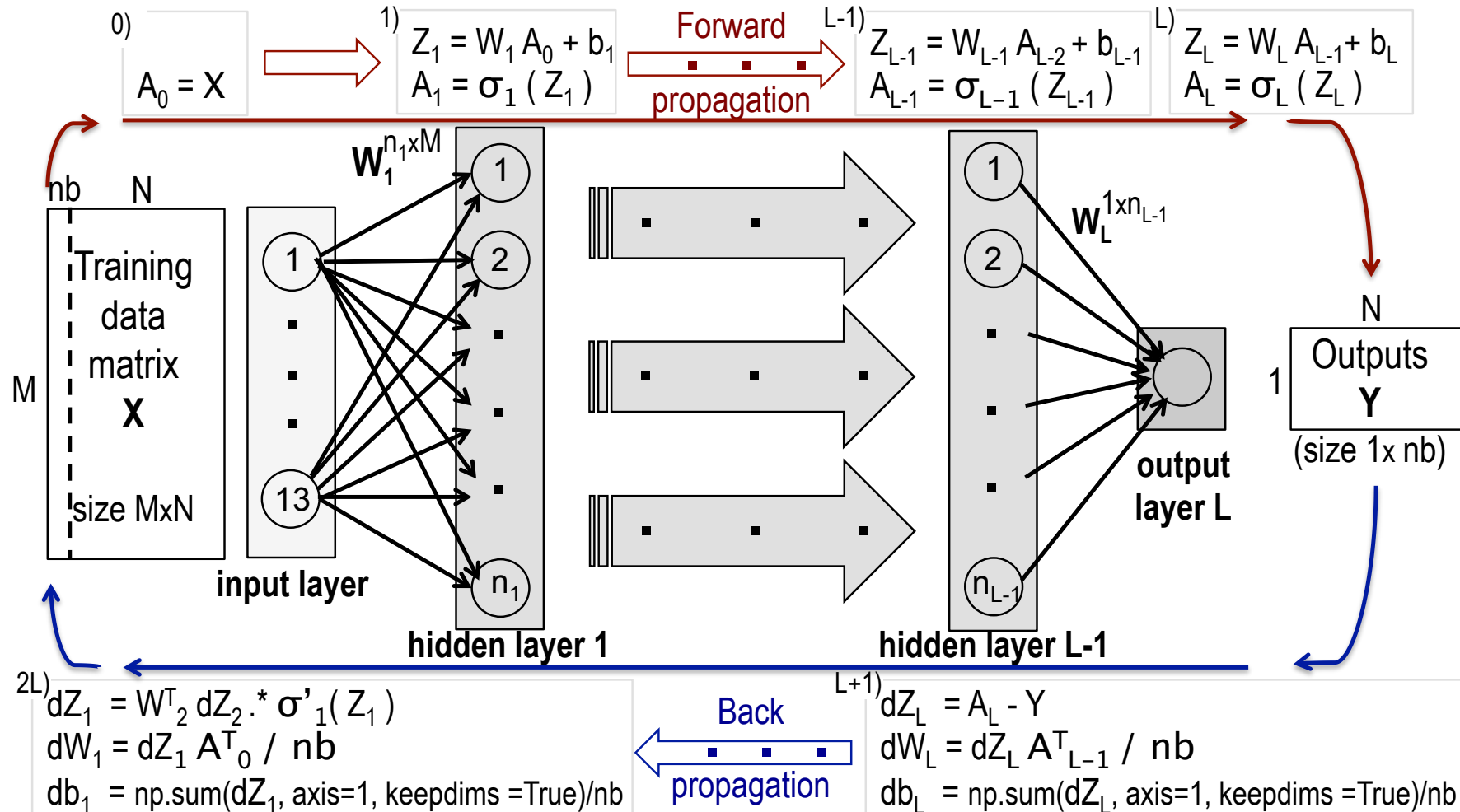


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



LA Operations in DNNs

- **Matrix-matrix (GEMM) multiplications**
For performance computations must be organized in terms of GEMMs:



LA Operations in DNNs

- **Matrix-matrix (GEMM) multiplications**
For performance computations must be organized in terms of GEMMs
- LA can be sparse, or dense, low-rank matrices, etc.
- **Batched LA**
- **Convolutions in Convolutional NNs (CNNs), with various ways to compute**
 - Directly using batched LA
 - Batched GEMMs
 - FFTs – convolutions $f * g$ of images f and filters g can be accelerated through FFT, as shown by the following equality, consequence of the convolution theorem:

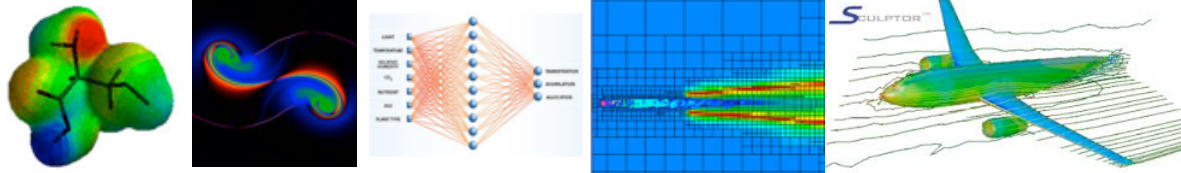
$$f * g = \text{FFT}^{-1} [\text{FFT}(f) .* \text{FFT}(g)],$$

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

- Winograd
- **Use of multi and mixed-precision calculations**

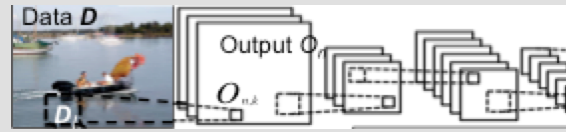
MagmaDNN

Applications



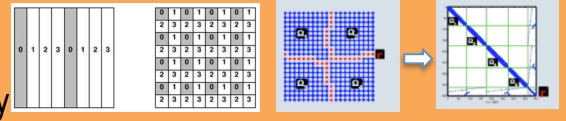
MagmaDNN

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

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

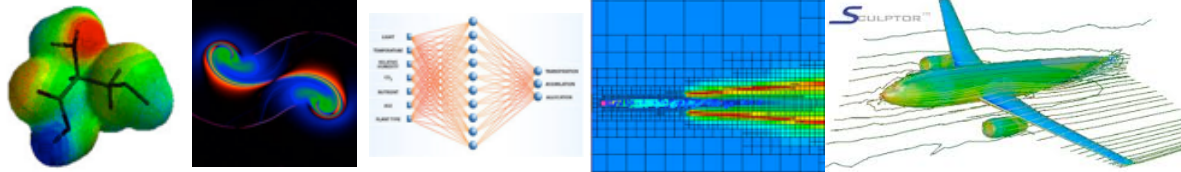
Run-time/comm. APIs

Vendor Libraries

- 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.

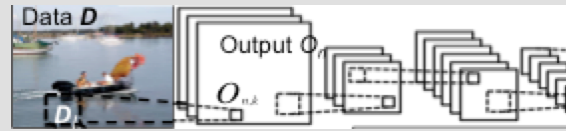
MagmaDNN

Applications



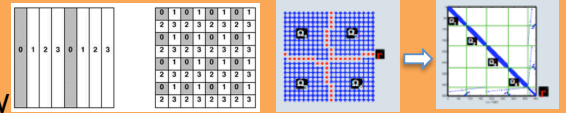
MagmaDNN

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

MAGMA (dense)

MAGMA Batched

MAGMA Sparse

Shared memory

BLAS API

LAPACK API

Batched BLAS API

OpenMP

MKL

ESSL

cuBLAS

ACML

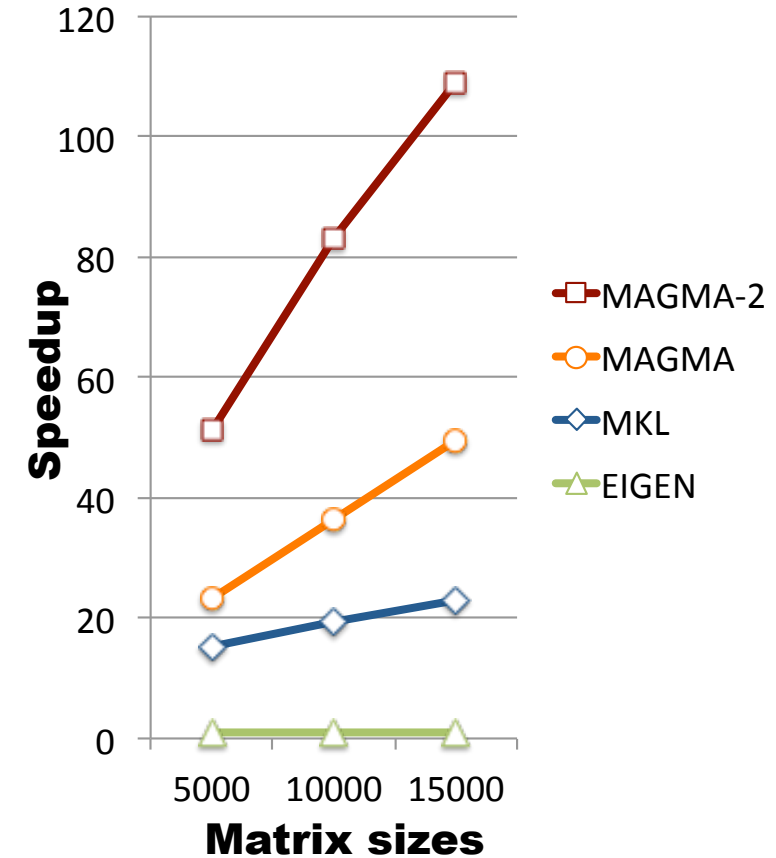
LA libraries

Standard LA APIs

Run-time/comm. APIs

Vendor Libraries

SVD performance speedup



Scaling **ML AI + HPC** simulations to Exascale

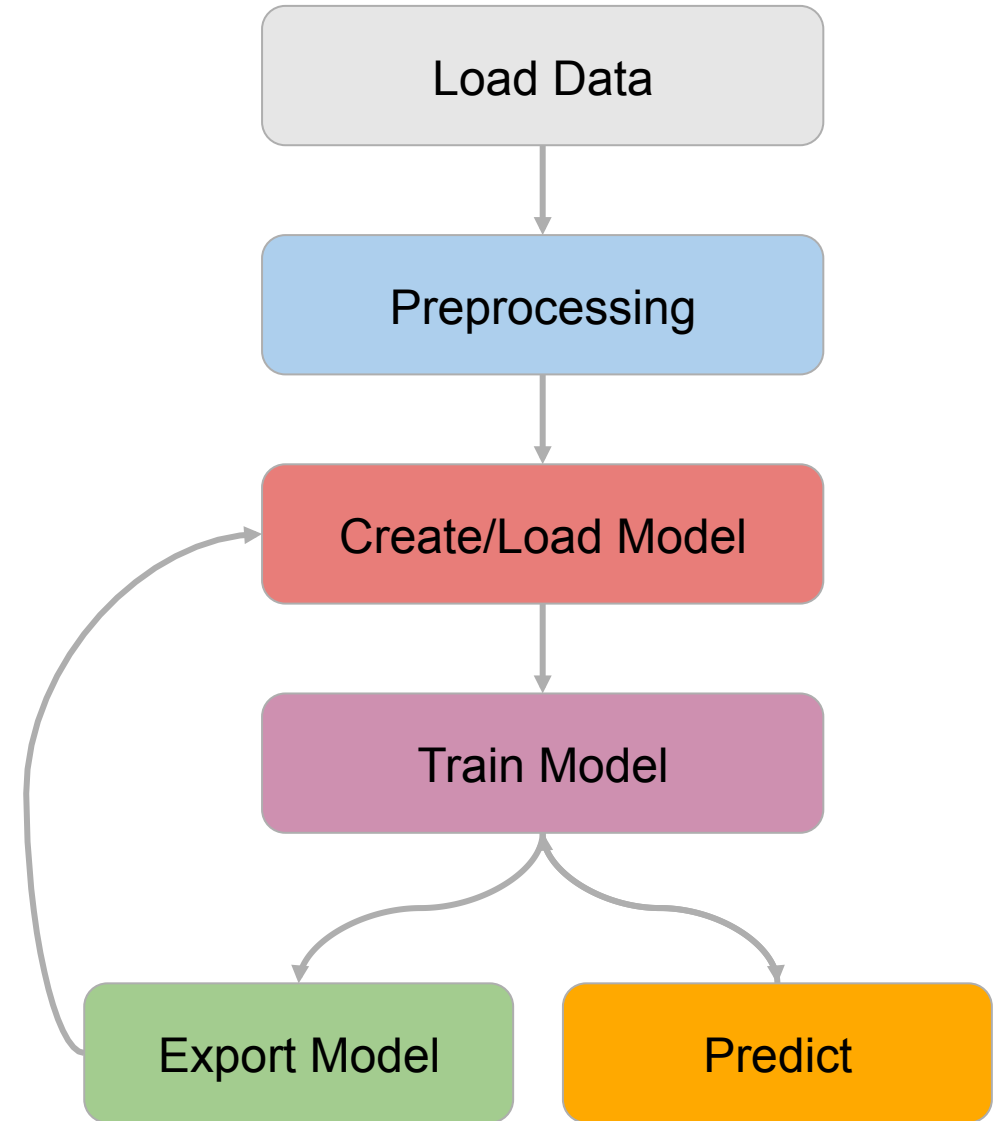
- **A position paper addressing challenges in integrating ML & traditional compute-intensive HPC simulations**

R. Archibald, E. Chow, E. D'Azevedo, J. Dongarra, M. Eisenbach, R. Febbo, F. Lopez, D. Nichols, S. Tomov, K. Wong, J. Yin, "Integrating Deep Learning in Domain Sciences at Exascale", Smokey Mountains Computational Sciences and Engineering Conference (SMC'2020), Aug. 26-28, 2020. (available at arXiv: 2011.11188)

- **Many ML/DL frameworks (TensorFlow, PyTorch, MxNet, etc.)**
 - developed by industry;
 - often targeting cloud environments for data-driven applications;
 - they are not necessarily suitable for scaling HPC simulations on large-scale supercomputers
- **To address the issues, we build a software infrastructure specifically for integrating ML and HPC simulations on petascale to exascale heterogeneous systems: DNN + Workflow**
 - 1) **Build a new C++ modular DNN framework, MagmaDNN, which is based on the LA software, MAGMA, for portability and scalability**
 - 2) **Provide a parallel workflow system to run combinations of ML & HPC codes**
 - 3) **Introduce algorithms for scheduling, auto tuning based and asynchronous solvers**
 - 4) **Discuss two DOE applications, materials science and climate data compression**

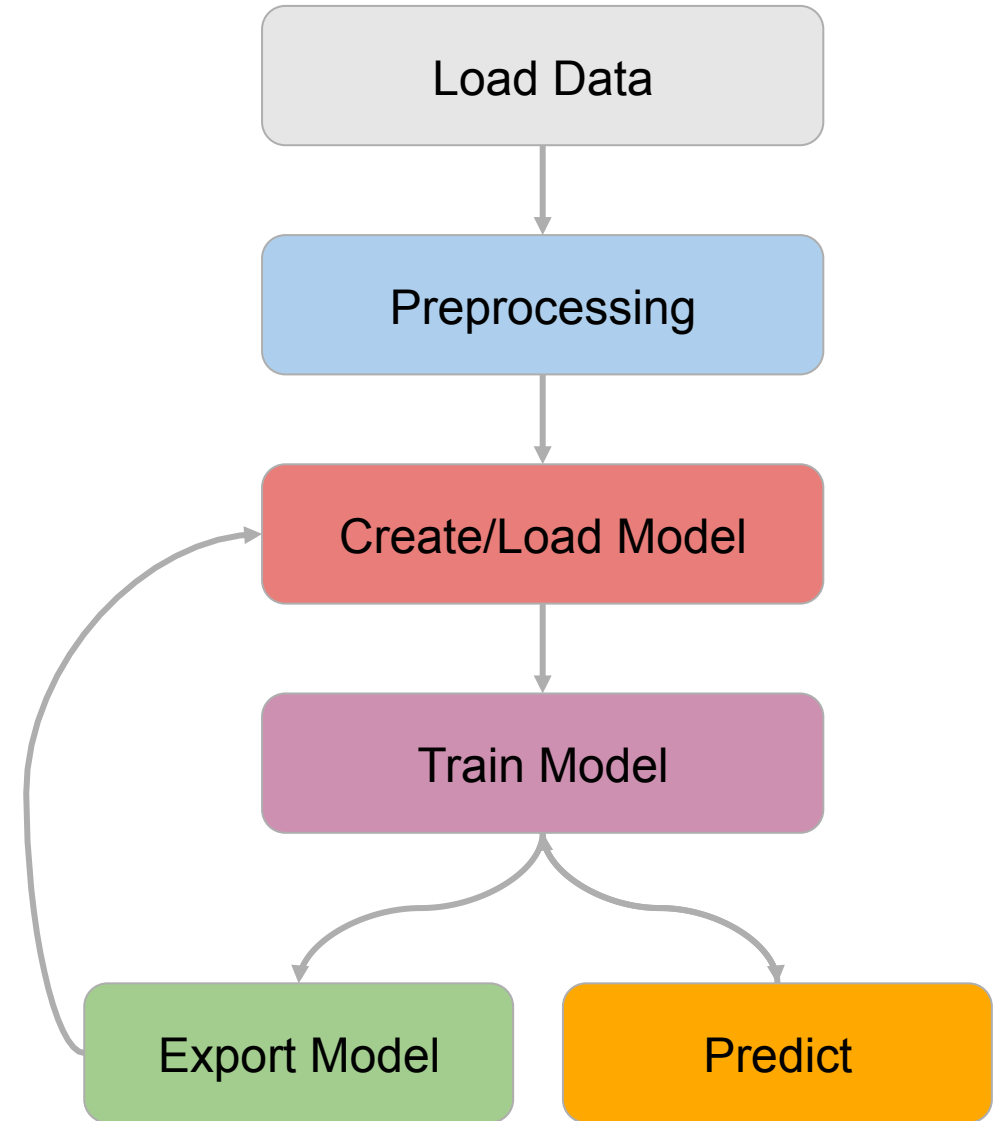
Design process

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

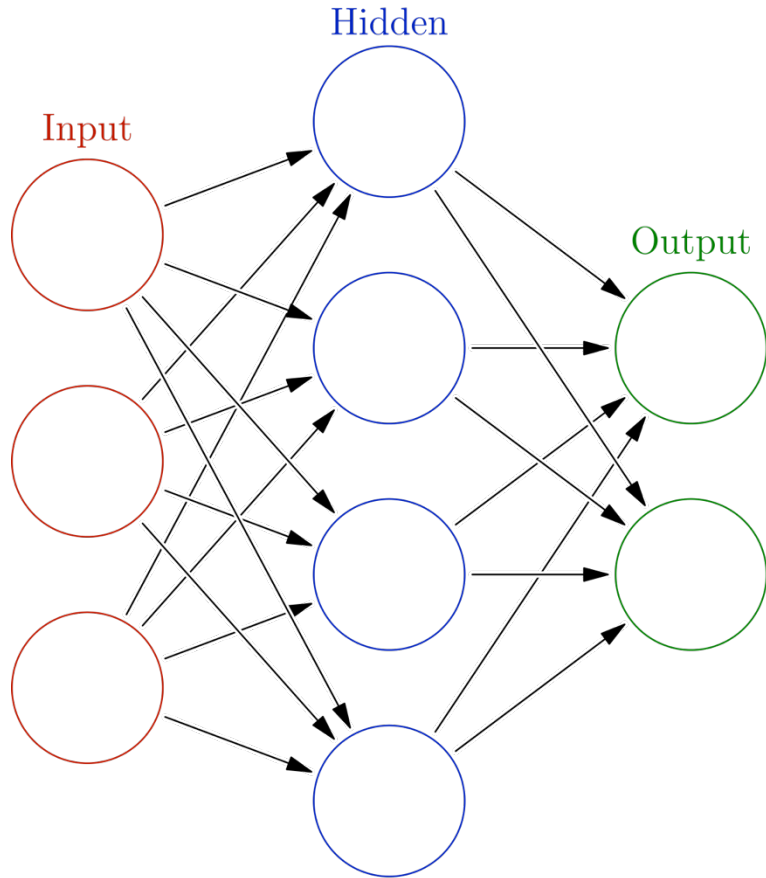


Workflow

- **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 .

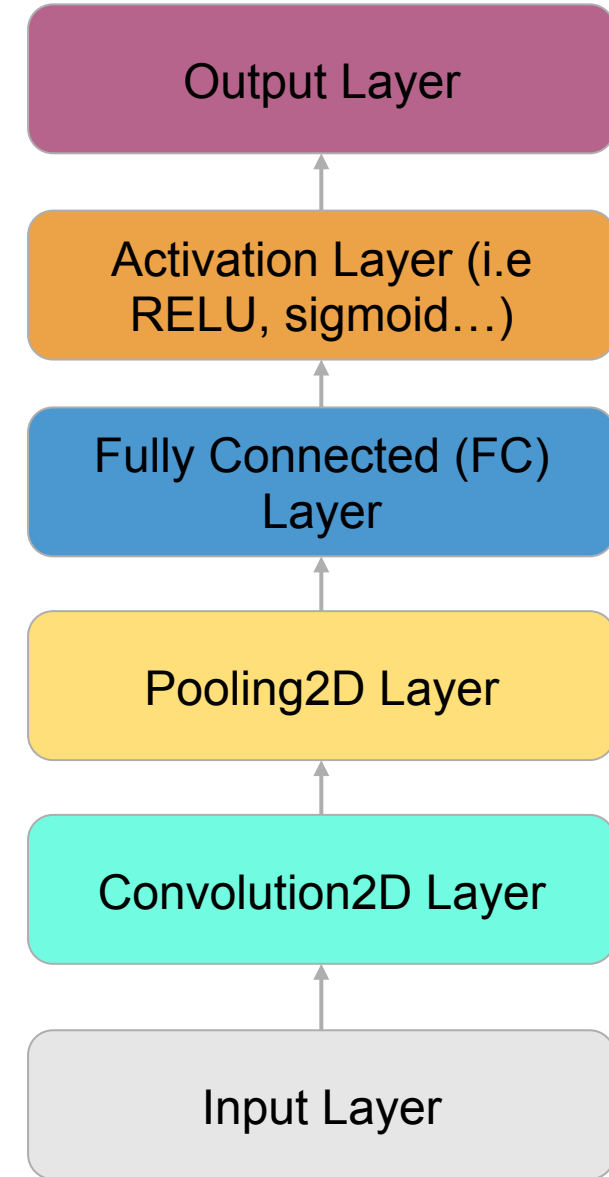
After optimizing some loss criterion w.r.t. the parameters of f , the function (or “network”) becomes an accurate predictor of highly abstracted data.

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

Neural Network Ideas (cont.)

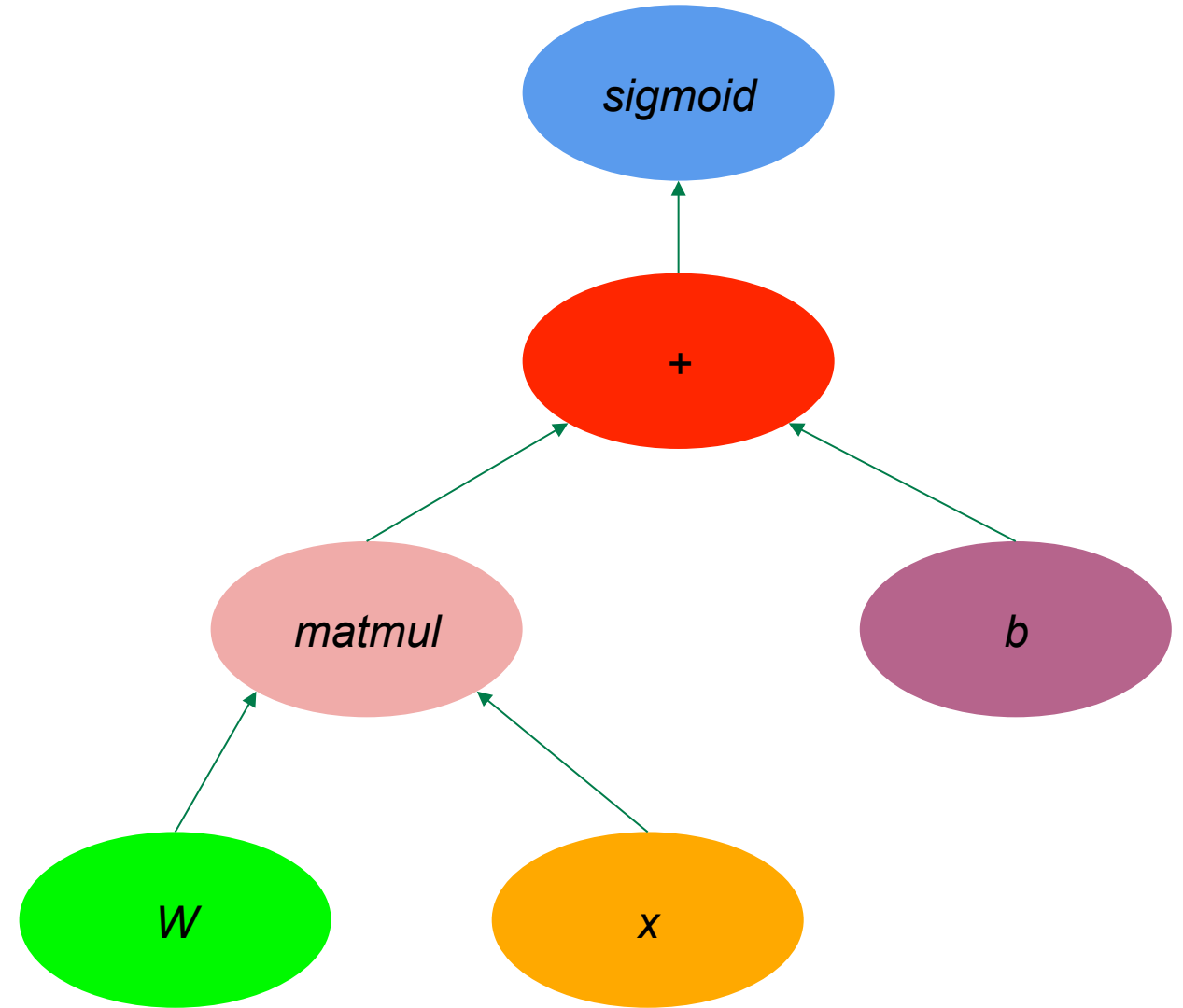
- Layers

- 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)



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)

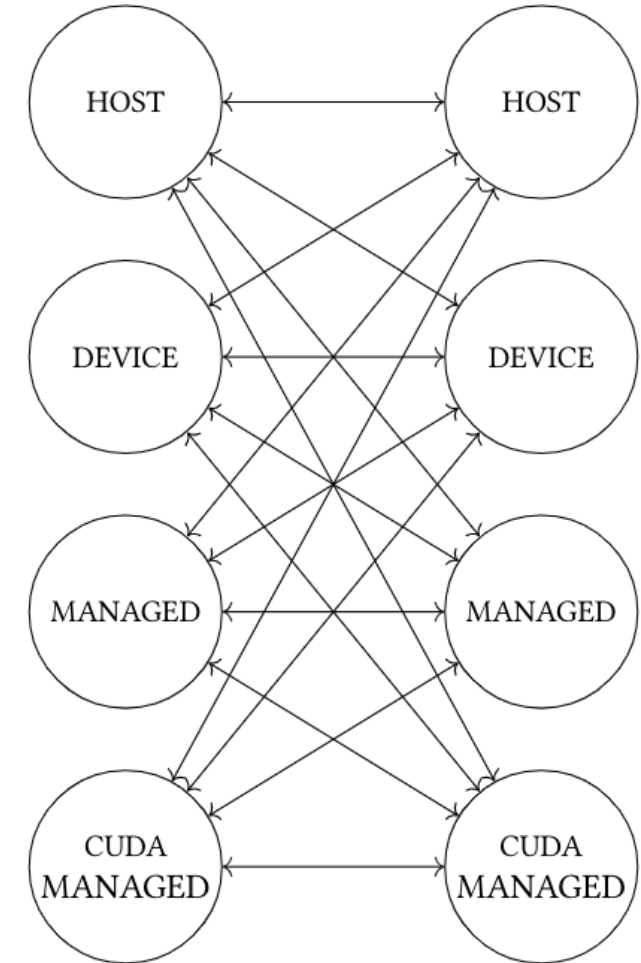
```
auto A = op::var<float> ("A", {4, 5}, {GLOROT, {1.5, 2.0}}, MANAGED);
auto X = op::var<float> ("X", {5, 4}, {UNIFORM, {0.0, 1.0}}, MANAGED);
auto B = op::var<float> ("B", {4, 4}, {DIAGONAL, {1, 2, 3, 4}}, MANAGED);

/* 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.

Scalar Vector Matrix Tensor

1

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

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();
```

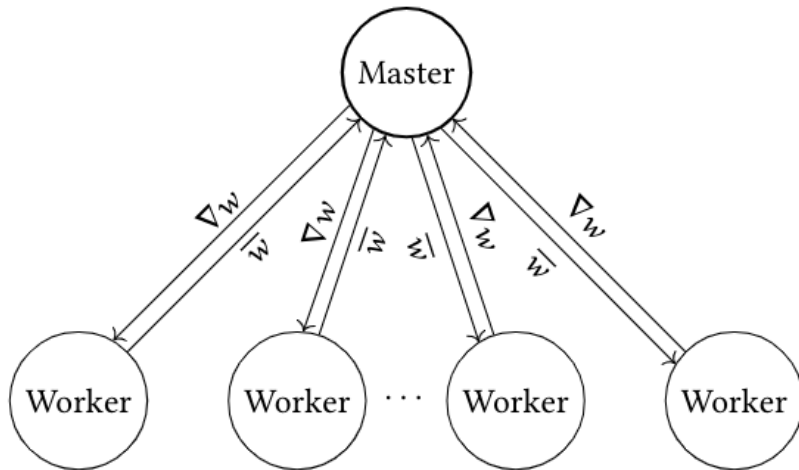
Training (example)

```
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);
```

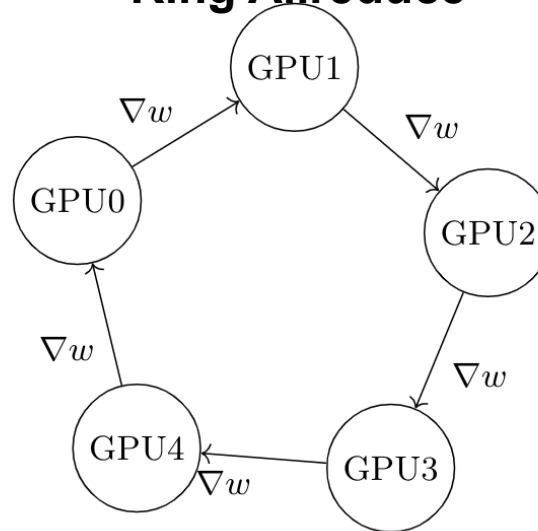
Distributed Training

- 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 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

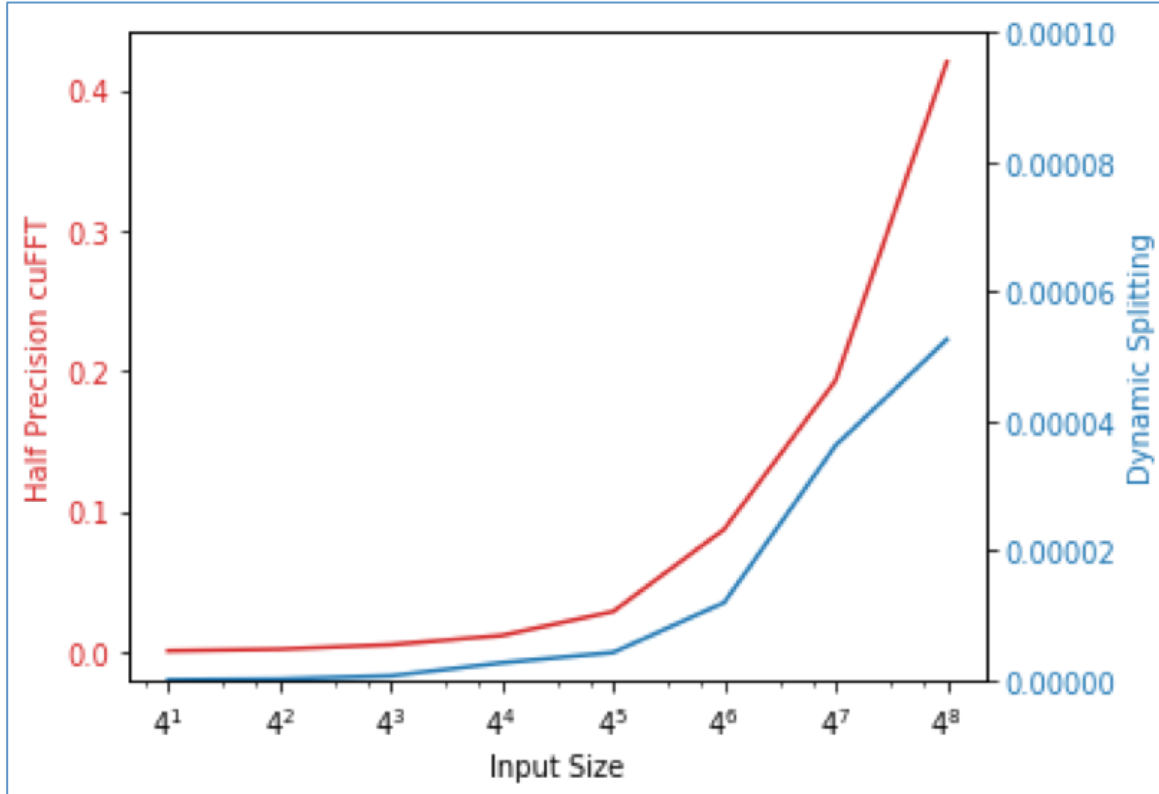
- Developed **mixed-precision (FP16-FP32) FFT** using the GPU's Tensor Cores (TC) acceleration
 - **Dynamic splitting to increase the FP16 accuracy, while using high-performance TC**

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

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

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

Reduced and Mixed Precision : scaling performance on new hardware



- ✓ **Key step** is expressing vector in FP32 as scaled sum of FP16 vectors
- ✓ $\mathbf{x}_{fp32}(:) = s1_{fp32} * \mathbf{x1}_{fp16}(:) + s2_{fp32} * \mathbf{x2}_{fp16}(:)$

Enhance performance for large scale image data applications

- ✓ Recursive Radix-4 or Radix-8 FFT implemented as batched matrix multiplication on tensor cores
- ✓ 4x4 Fourier matrices are exactly representable in FP16, rational approximation for 8x8 Fourier matrices
- ✓ 2D FFT and 3D FFT implemented as batched 1D FFT
- ✓ This approach has much higher accuracy compared to FFT of FP16

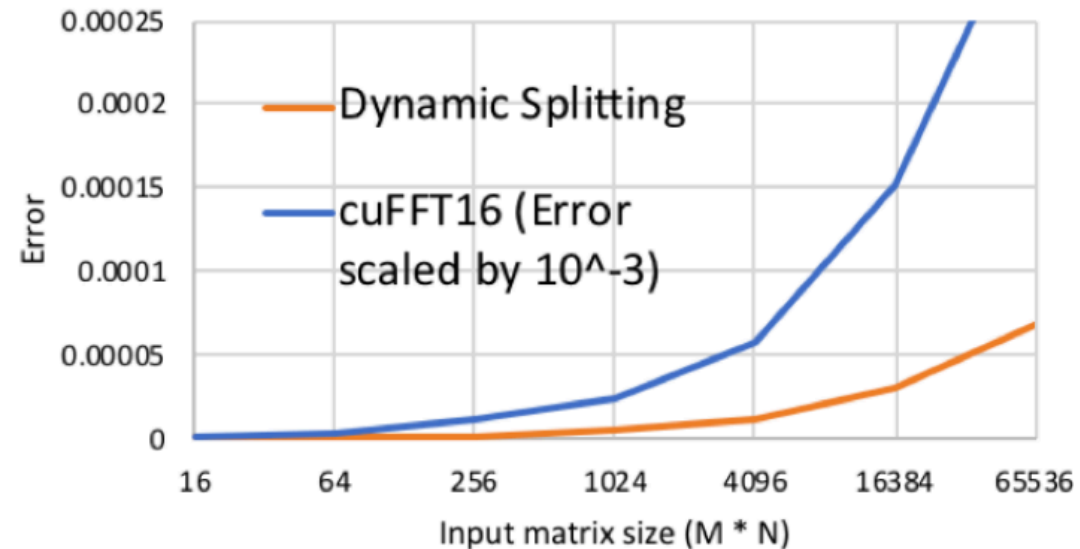
Accuracy of the mixed-precision (FP16-FP32) FFT

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

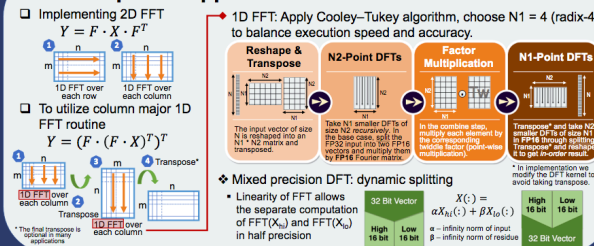
Overview

- 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

- The method preserves high accuracy, even with growing matrix sizes
 - The cost of dynamic splitting and combine is not significant
-
- | Input Matrix Size (M * N) | GEMM (ms) | Splitting (ms) | Combine (ms) |
|---------------------------|-----------|----------------|--------------|
| 16 | ~100 | ~10 | ~10 |
| 64 | ~400 | ~20 | ~20 |
| 256 | ~1600 | ~40 | ~40 |
| 1024 | ~6400 | ~80 | ~80 |
| 4096 | ~25600 | ~160 | ~160 |
| 16384 | ~102400 | ~320 | ~320 |
| 65536 | ~409600 | ~640 | ~640 |
- The relative error of 2D FFT at different input sizes (horizontal dimension * vertical dimension), using our implementation and half precision cuFFT.
- | Implementation | Data range | $\pm 10^{-7}$ | $\pm 10^{-4}$ | $\pm 10^{-1}$ | $\pm 10^2$ | $\pm 10^4$ | $\pm 10^6$ |
|----------------|------------|---------------|---------------|---------------|------------|------------|------------|
| cuFFT16 | 59.088 | 5.818% | 5.602% | 5.783% | N/A | N/A | N/A |
| Splitting | 0.002% | 0.001% | 0.001% | 0.001% | 0.001% | 0.001% | 0.001% |
- The implementation can handle a wide range of inputs and produce accurate 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. Redwood City: Benjamin Cummings, 1994.

[2] Stear, Peter, et al. "Taking a quantum leap in time to solution for simulations of high-temperature superconductors." *International Journal of High Performance Computing, Networking, Storage and Analysis* (SC), 2013 International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2013.

[3] Göddelke, Dominik, Robert Strzodka, and Stefan Turek. "Performance and accuracy of real, emulated and mixed-precision solvers in FEU simulations." *International Journal of High Performance Computing, Networking, Storage and Analysis* (SC), 2013 International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2013.

[4] Appleby 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**

...

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>

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.

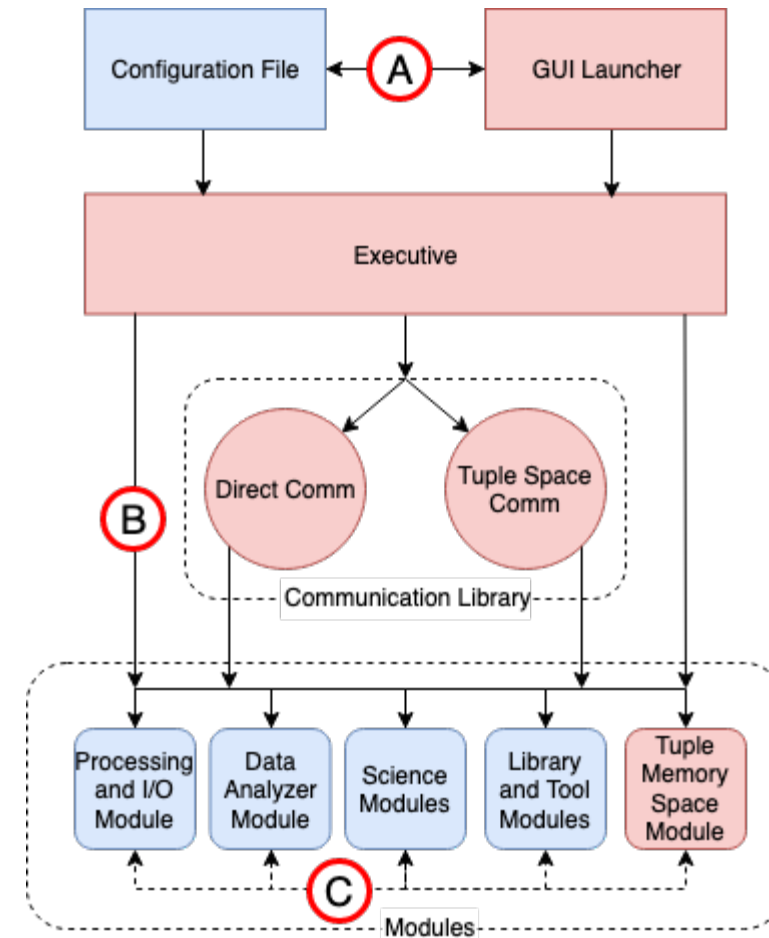
Hyperparameter optimization

openDIEL is designed to scaling compute and data intensive applications

- ✓ www.bitbucket.org/cfdl/opendiel
- ✓ An open source light weight parallel workflow framework designed to scale and run inter-disciplinary simulations on large-scale heterogeneous HPC platforms.
- ✓ Schedule and run a collection of user's codes (scripts, serial, and parallel programs) under a single MPI executable, similar to swift-T, is ideal for combining compute-intensive and data-driven applications
- ✓ Provide two communication interfaces, a direct one to one communicator and a asynchronous tuple space communicator, for exchanging data among different programs.
- ✓ Show workflows for materials sciences and climate applications

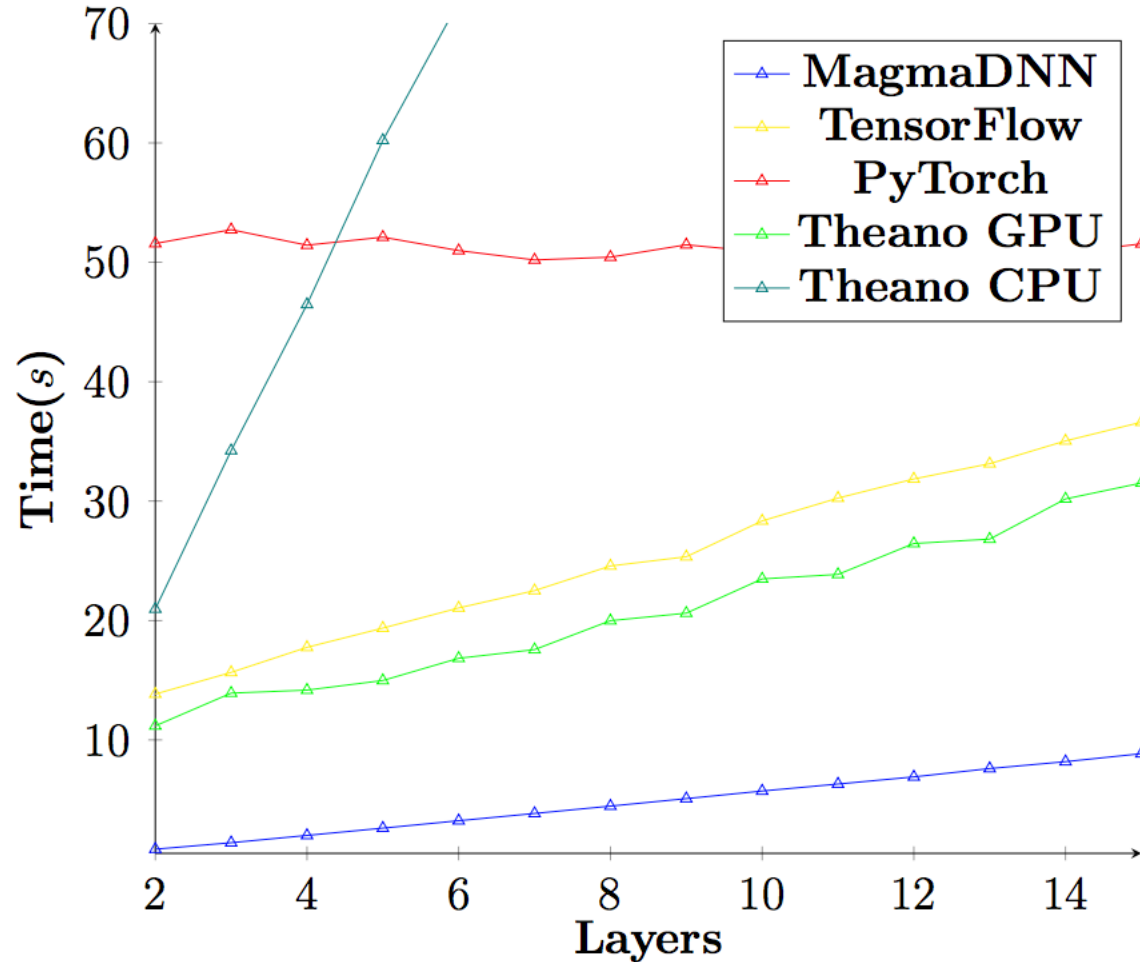
OpenDIEL architecture:

- (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)

MLP Time Comparison on MNIST



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
#Hidden Units Layer	528

MagmaDNN scalability and SGD speedup

Asynchronous Stochastic Gradient Descent

Asynchronous SGD compared to the classical synchronous SGD:

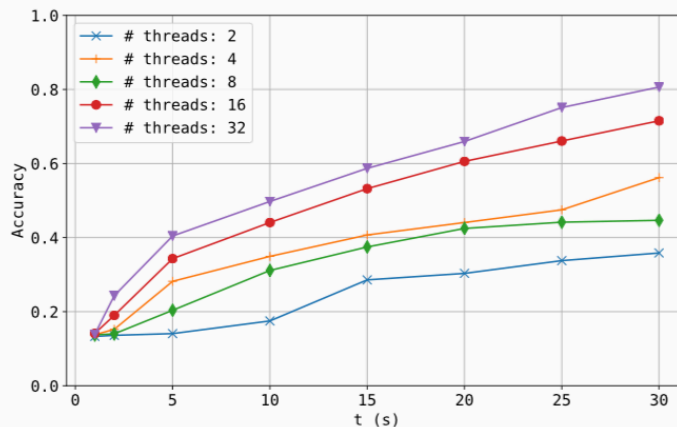
- ▲ Little to no synchronization or locking mechanism \Rightarrow better scalability
- ▼ Delayed gradient update \Rightarrow slower convergence.

Examples of ASGD algorithms:

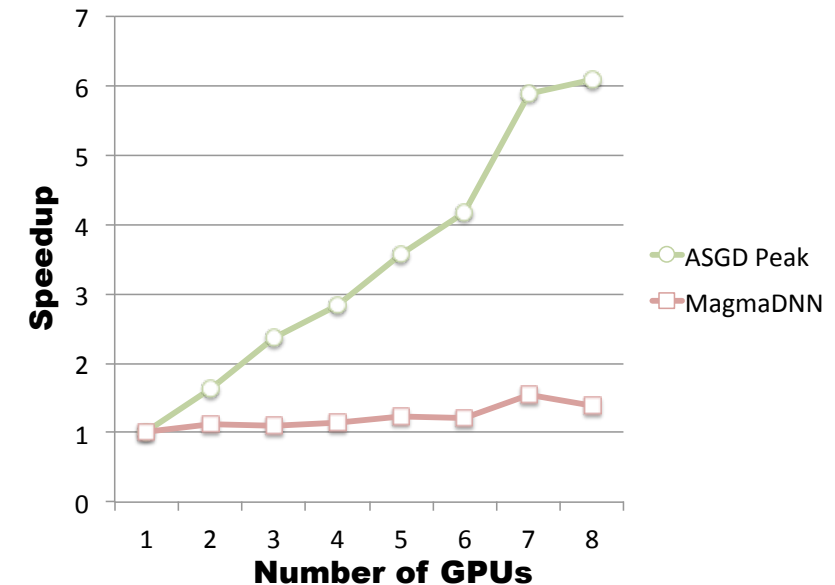
- Shared-memory architectures: **Hogwild!**.
- Distributed-memory architectures: **Downpour SGD**.

ASGD in **MagmaDNN**:

- Master worker: performs SGD iterations on global parameter.
- Slave workers: Compute local gradients.

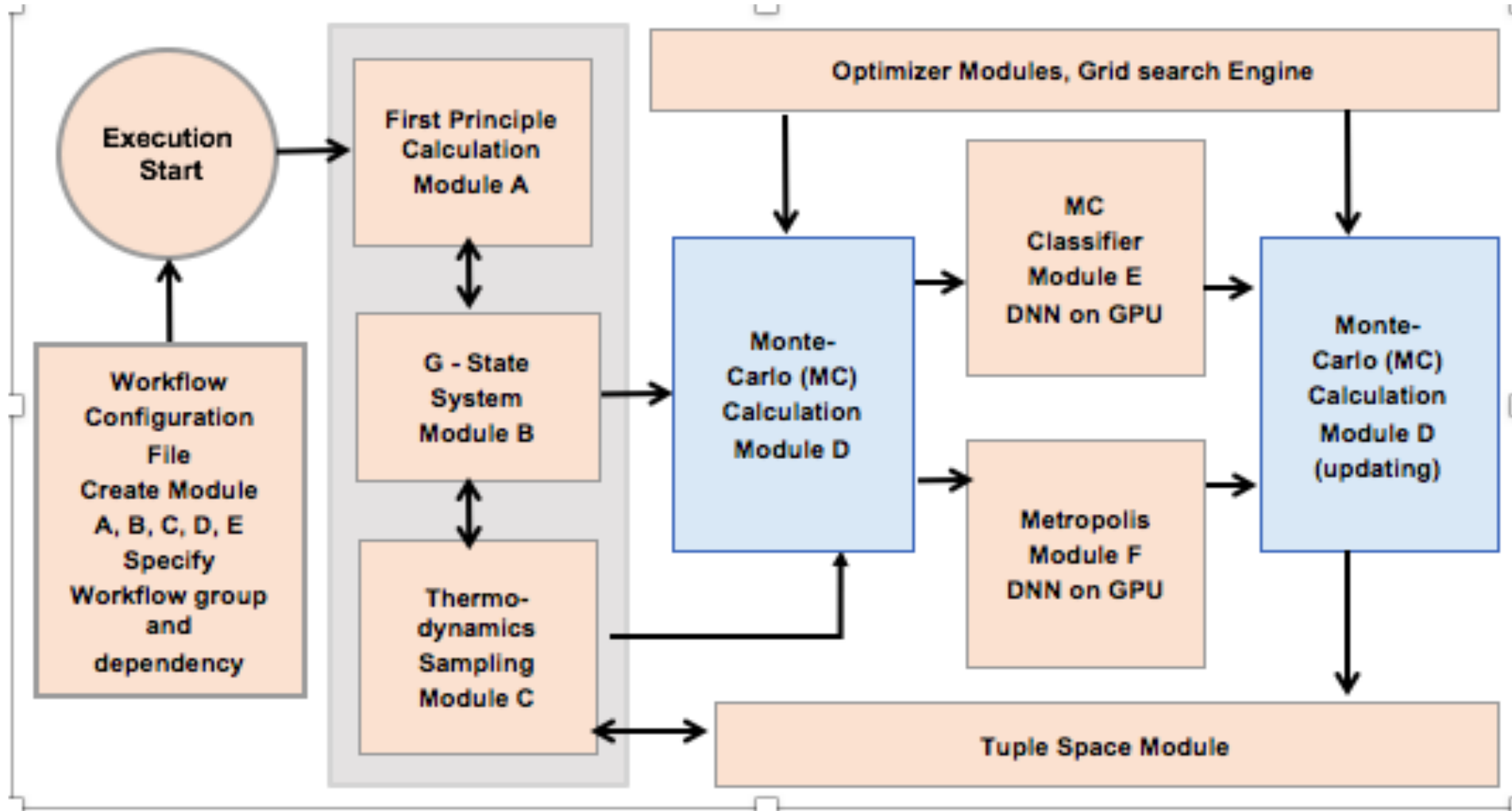


Speedup vs. TensorFlow



Applications in Materials Science and Microscopy

Using openDIEL to combine DFT, ML, MC on exascale platform

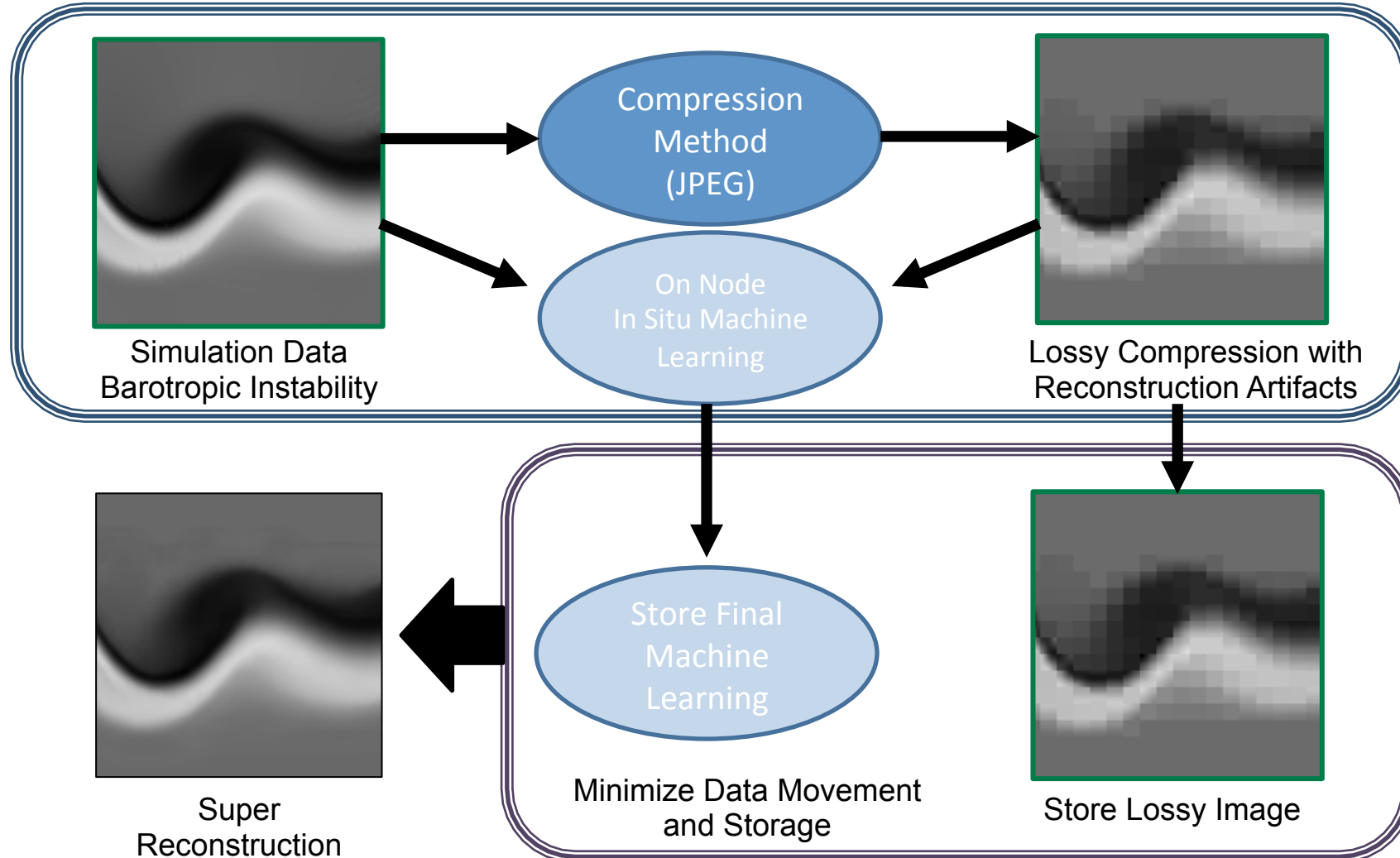


Tight coupling of first principles statistical mechanics of materials to calculate the temperature dependence of materials - requires the calculations of many possible atomic configurations within these materials using a Monte Carlo approach, where the probability of the individual states would be evaluated using an expensive density functional theory calculation.

Train a surrogate model that can replace the expensive (in the order of multiple node hours per data point) first principles calculation within the Monte Carlo sampling of possible configurations.

Image Compression for Climate Science Simulation

Workflow to perform in-situ DL training for both compressed and original images



- it is possible to train in situ networks (RNNs) to reduce the error in lossy compression—obtaining multiple orders of improvement.
- Going forward, it will be necessary to further improve in situ compression and analysis in order to maximize discovery with HPC simulations

MagmaDNN benchmarks and testing examples ...



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 EEG, we can detect the user's intention by classifying their EEG signals. EEG signals which recorded electrical activities of the brain. By using advanced machine learning technology, we can control advanced prosthetic devices. Patients can be benefited from this technology.



Ob

- To classify the user intention from the EEG 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)



INT

There are three known basic modes which is a 2688 by 2688 input image I , we try to decompose it into three basic modes. It is closely represented as a sum of three basic modes, namely

$$I = I_1 + I_2 + I_3$$

The problem can be solved by the least squares method. However, this 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 network 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.

Accelerating FFT with half-precision floating point hardware on GPU

Anumeena Sorna (NITT) & Xiaohe Cheng (HKUST)




Mentor: Eduardo D'Azevedo (ORNL) & Kwai Wong (UTK)

Abstract

We present a method for accelerating the computation of the Discrete Fourier Transform (DFT) on GPU using half-precision floating point hardware. The DFT is computed using a single precision algorithm by recombining the results of the half-precision DFT algorithm. We hope to find a more specific computation.

Discrete Fourier Transform

The DFT converts time-domain signals according to the following equation:

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

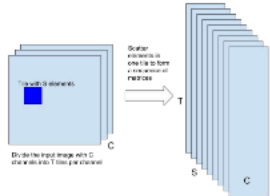
Students: Alex Gessinger(SRU), Sihan Chen(CUHK)

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

Current work and Future directions

- **Development in AI software and tools that scale well on Exascale systems is important, and even more critical for AI frameworks that also work well across the existing spectrum of exascale applications**
- **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.
 - 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**

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



U.S. DEPARTMENT OF
ENERGY



CEED: Center for
Efficient Exascale Discretizations



THE UNIVERSITY of
TENNESSEE
Department of Electrical Engineering
and Computer Science