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

Lucien Ng
The Chinese University of Hong Kong

Kwai Wong
The Joint Institute for Computational Sciences (JICS), UTK and ORNL

Azzam Haidar, Stanimire Tomov, and Jack Dongarra
(haidar | tomov | dongarra@icl.utk.edu)
The Innovative Computing Laboratory, UTK

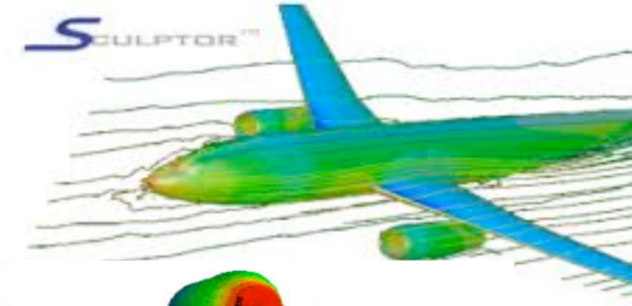
2017 Summer Research Experiences for Undergraduate (REU)
Research Experiences in Computational Science, Engineering, and Mathematics (RECSEM)
Knoxville, TN

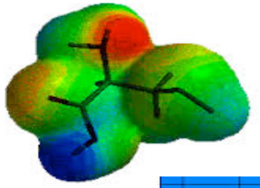
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

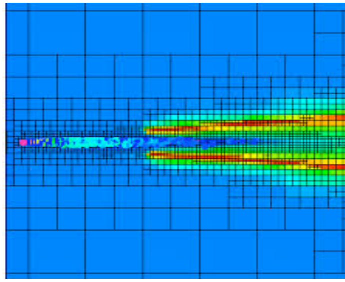
Dense Linear Algebra in Applications

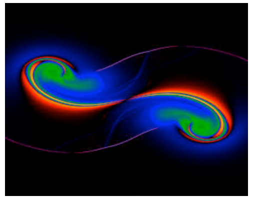
Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:

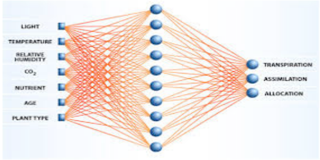
- **Linear systems:** Solve $Ax = b$
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** Find x to minimize $\| Ax – b \|$
  - Computational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











Dense Linear Algebra in Applications

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

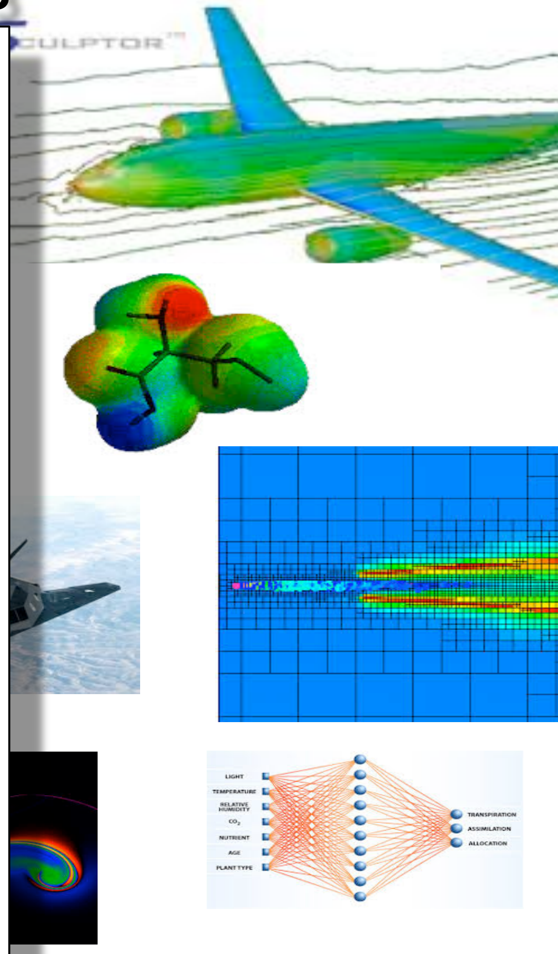
- **Linear systems:** Solve Ax = b
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- **Least squares:** Find x to minimize || Ax – b ||
  - Computational statistics (e.g., linear least squares or ordinary least squares), econometrics, control theory, signal processing, curve fitting, and many more
- **Eigenproblems:** Solve Ax = λ 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 Σ V* (Au = σv and A*v = σu)
  - 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 2.3

FEATURES AND SUPPORT

- MAGMA 2.3 FOR CUDA
- clMAGMA 1.4 FOR OpenCL
- MAGMA MIC 1.4 FOR Intel Xeon Phi

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



MAGMA

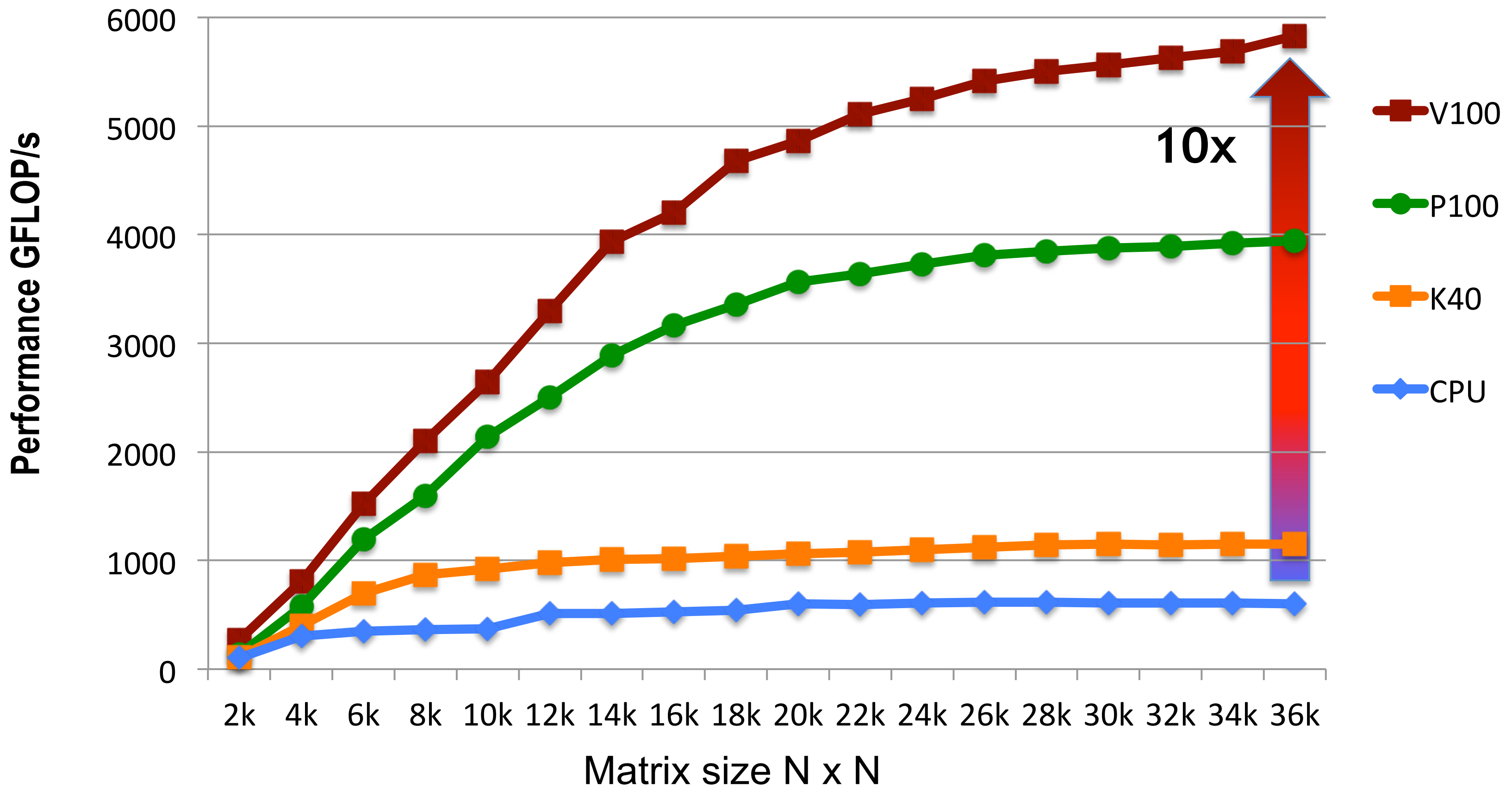
http://icl.cs.utk.edu/magma
https://bitbucket.org/icl/magma

Why use GPUs in HPC?

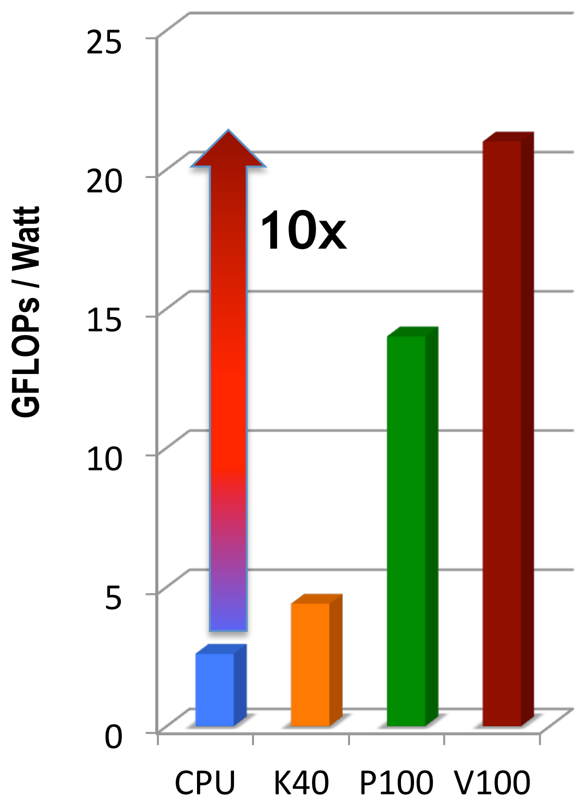
PERFORMANCE & ENERGY EFFICIENCY

MAGMA 2.3 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



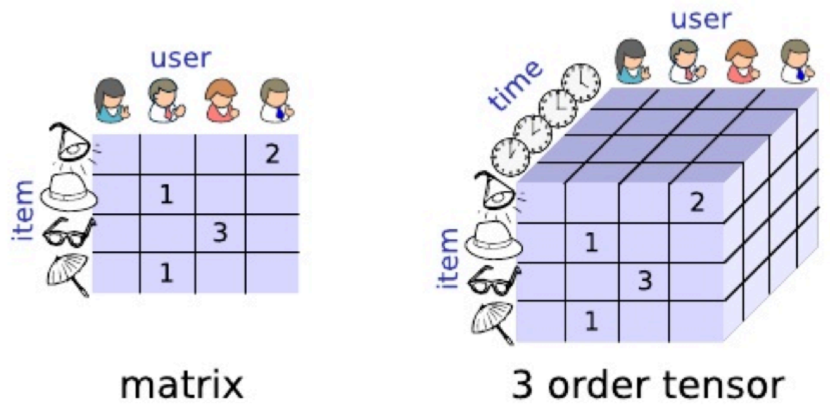
Energy efficiency
(under ~ the same power draw)



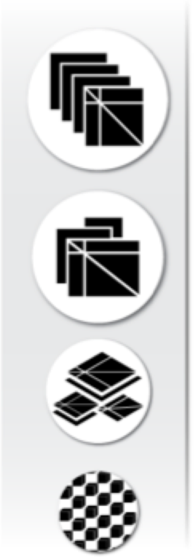
What about accelerated LA for Data Analytics?

- 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

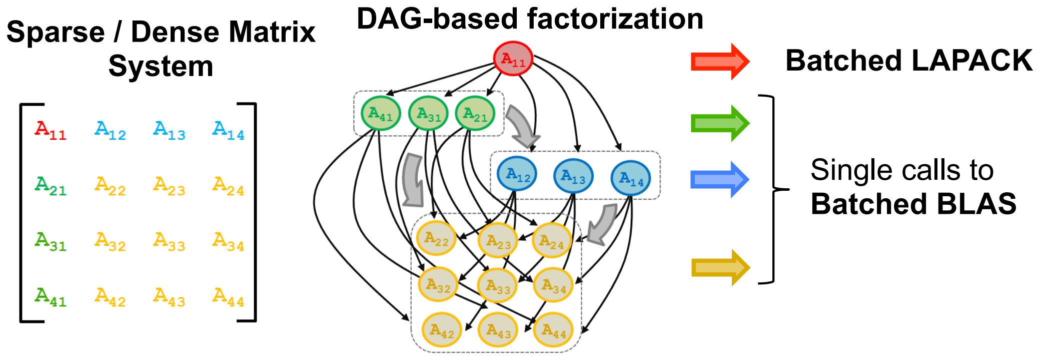
Tensors

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



Applications using high-order FEM

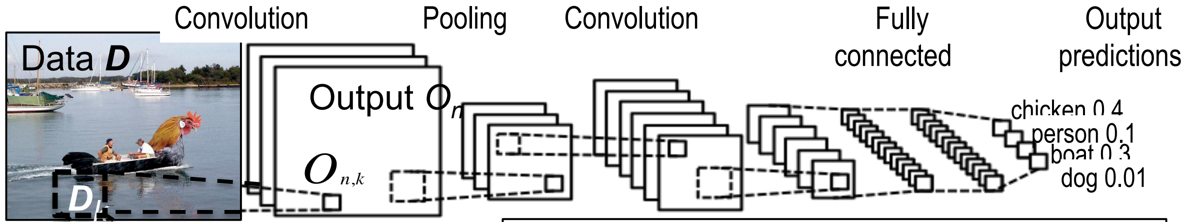
- Matrix-free basis evaluation needs efficient tensor contractions,

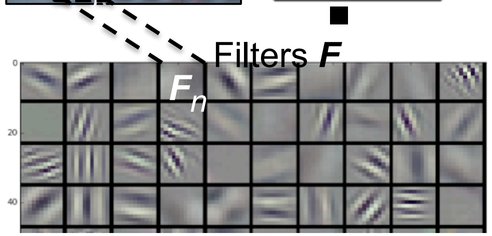
$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

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

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

Machine learning





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)

MagmaDNN – Data Analytics Tool

- **MagmaDNN 0.1-Alpha – 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

- **Implemented/proposed so far**
  - Tensors and tensor operations
  - Deep learning primitives:
    Fully-connected layers, convolutional layers, pooling layers, activation layers, and output layers.
    All of them support SGD back-propagation training
  - Established adapters for calling CuDNN
  - Applied MagmaDNN to the MNIST benchmark using multilayer perceptron or a convolutional neural network.

Provided in **MAGMA 2.3**

FEATURES AND SUPPORT

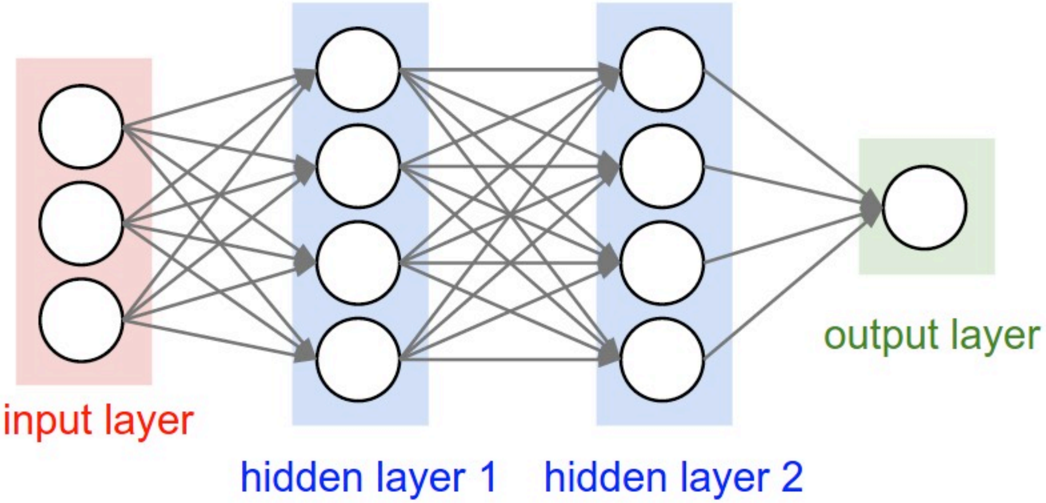
- **MAGMA 2.3** FOR CUDA
- **clMAGMA 1.4** FOR OpenCL
- **MAGMA MIC 1.4** FOR Intel Xeon Phi

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

MAGMA http://icl.cs.utk.edu/magma **https://bitbucket.org/icl/magmadnn**

Fully connected layers

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

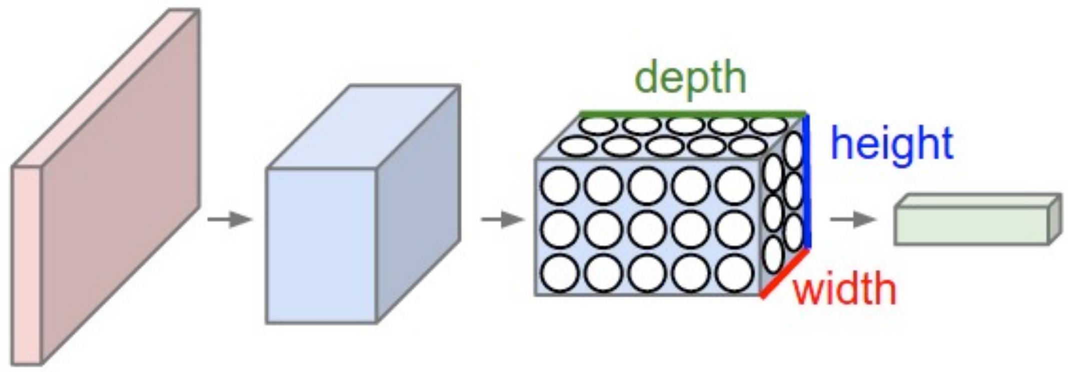
```
// Create layers for the network
FCLayer<float> *FC1          = new FCLayer<float>(&inputLayer, 128);
ActivationLayer<float> *actv1 = new ActivationLayer<float>(FC1, SIGMOID);
FCLayer<float> *FC2          = new FCLayer<float>(actv1, n_output_classes);
```

- **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

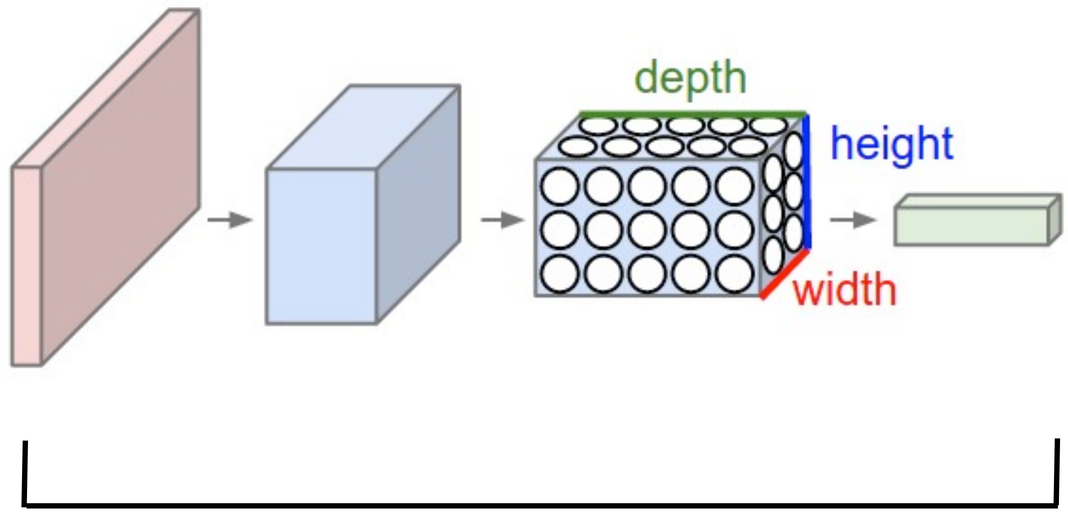
Convolution Network (ConvNet) example



- 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

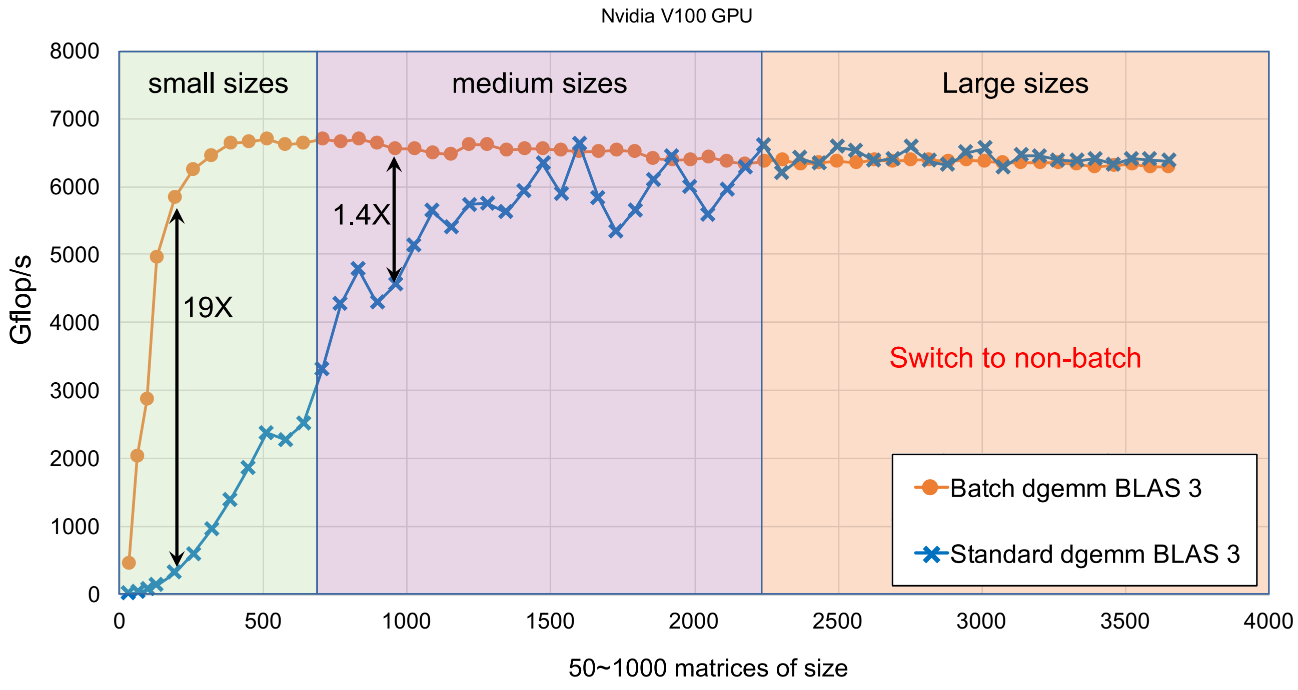
- **Convolutions can be accelerated in various ways:**
  - **Unfold and GEMM**
  - **FFT**
  - **Winograd minimal filtering – reduction to batched GEMMs**

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**

How to implement fast batched DLA?

Problem sizes influence algorithms & optimization techniques

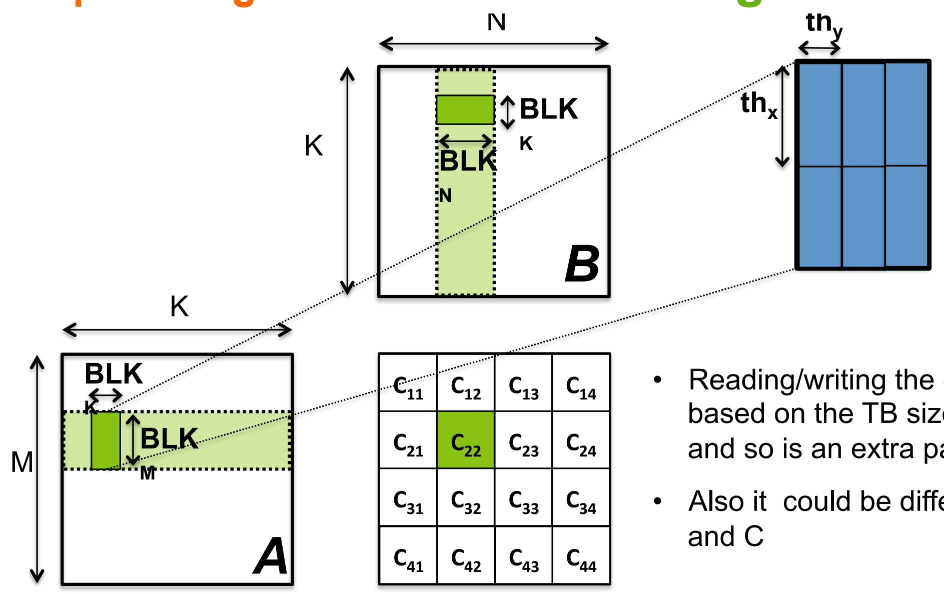


Matrix sizes (fixed) in the batch

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

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

Optimizing GEMM’s: Kernel design

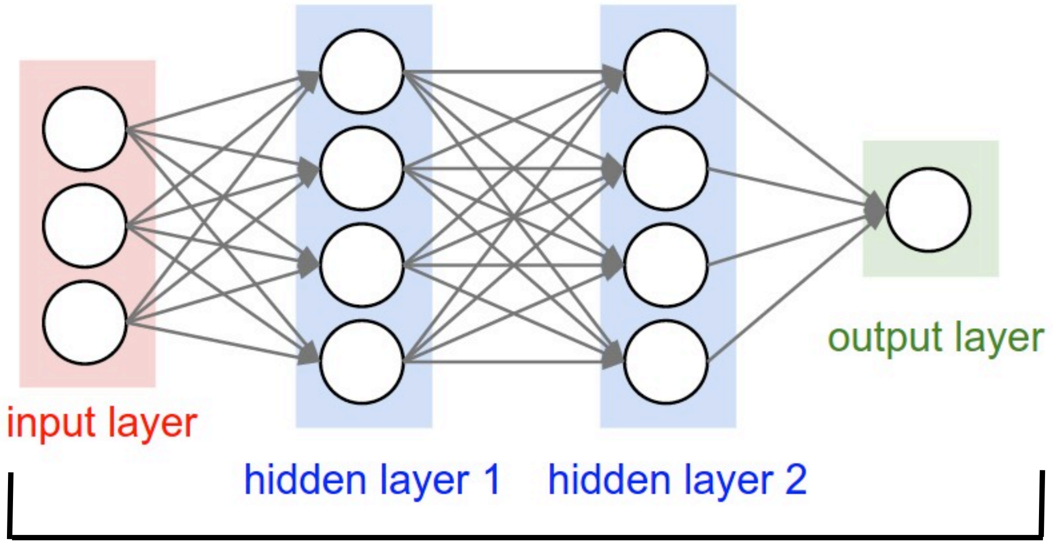


- Reading/writing the elements is based on the TB size (# threads) and so is an extra parameter.
- Also it could be different for A, B and C

Examples

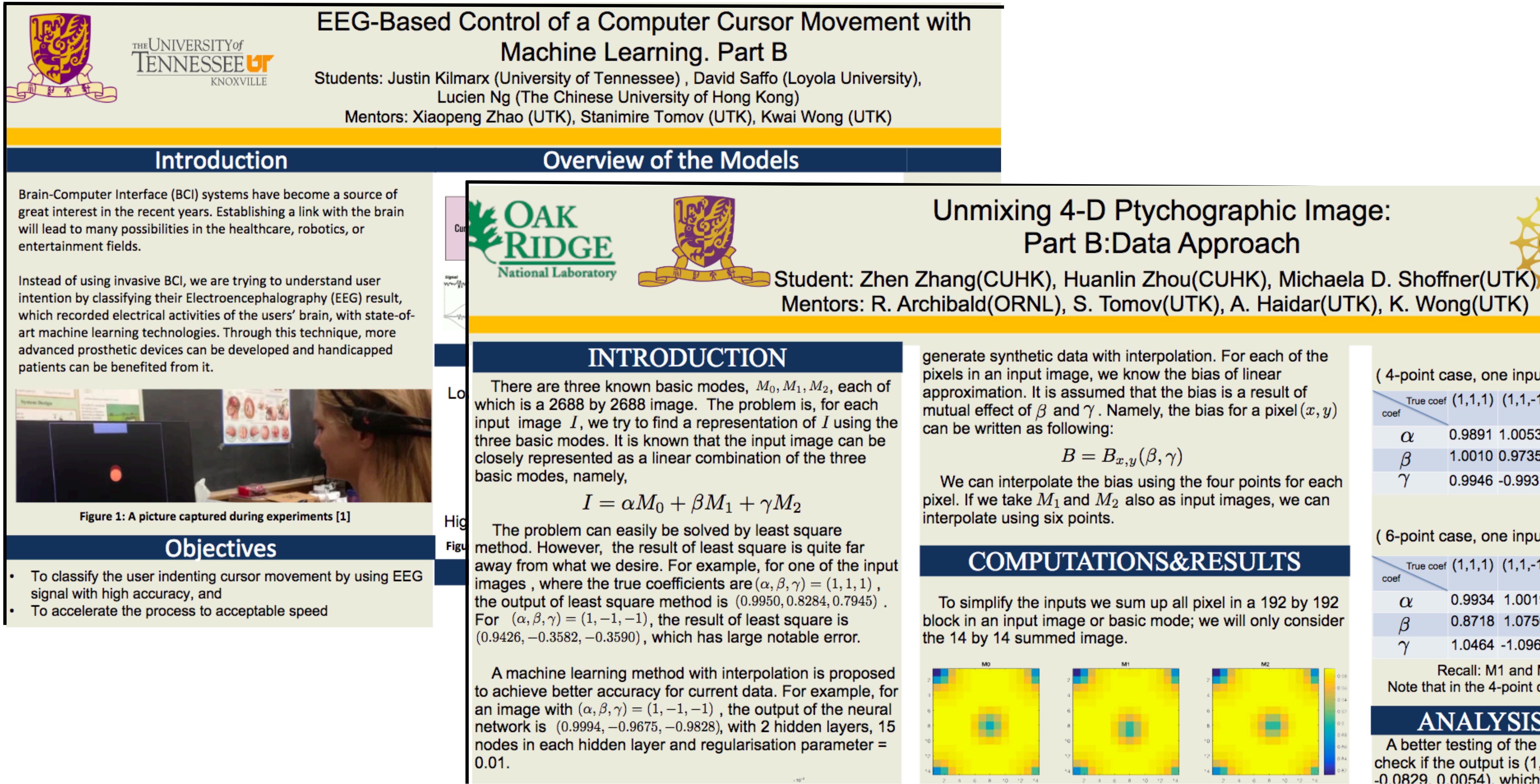
- MagmaDNN has testing/example drivers
- Example implementing the MNIST benchmark using MagmaDNN multilayer perceptron or a convolutional neural network

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

Examples …



THE UNIVERSITY of TENNESSEE UT 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)
Introduction
Overview of the Models
Brain-Computer Interface (BCI) systems have become a source of great interest in the recent years. Establishing a link with the brain will lead to many possibilities in the healthcare, robotics, or entertainment fields.
Instead of using invasive BCI, we are trying to understand user intention by classifying their Electroencephalography (EEG) result, which recorded electrical activities of the users' brain, with state-of-art machine learning technologies. Through this technique, more advanced prosthetic devices can be developed and handicapped patients can be benefited from it.
Figure 1: A picture captured during experiments [1]
Objectives
To classify the user indenting cursor movement by using EEG signal with high accuracy, and
To accelerate the process to acceptable speed
OAK RIDGE National Laboratory
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)
INTRODUCTION
There are three known basic modes, M_0, M_1, M_2, each of which is a 2688 by 2688 image. The problem is, for each input image I, we try to find a representation of I using the three basic modes. It is known that the input image can be closely represented as a linear combination of the three basic modes, namely,
$I = \alpha M_0 + \beta M_1 + \gamma M_2$
The problem can easily be solved by least square method. However, the result of least square is quite far away from what we desire. For example, for one of the input images , where the true coefficients are $(\alpha, \beta, \gamma) = (1, 1, 1)$, the output of least square method is $(0.9950, 0.8284, 0.7945)$. For $(\alpha, \beta, \gamma) = (1, -1, -1)$, the result of least square is $(0.9426, -0.3582, -0.3590)$, which has large notable error.
A machine learning method with interpolation is proposed to achieve better accuracy for current data. For example, for an image with $(\alpha, \beta, \gamma) = (1, -1, -1)$, the output of the neural network is $(0.9994, -0.9675, -0.9828)$, with 2 hidden layers, 15 nodes in each hidden layer and regularisation parameter = 0.01.
generate synthetic data with interpolation. For each of the pixels in an input image, we know the bias of linear approximation. It is assumed that the bias is a result of mutual effect of β and γ . Namely, the bias for a pixel (x, y) can be written as following:
$B = B_{x,y}(\beta, \gamma)$
We can interpolate the bias using the four points for each pixel. If we take M_1 and M_2 also as input images, we can interpolate using six points.
COMPUTATIONS&RESULTS
To simplify the inputs we sum up all pixel in a 192 by 192 block in an input image or basic mode; we will only consider the 14 by 14 summed image.
M0
M1
M2
(4-point case, one inpu
True coef (1,1,1) (1,1,-1
coef
α 0.9891 1.0053
β 1.0010 0.9735
γ 0.9946 -0.993
(6-point case, one inpu
True coef (1,1,1) (1,1,-
coef
α 0.9934 1.001
β 0.8718 1.075
γ 1.0464 -1.096
Recall: M1 and
Note that in the 4-point
ANALYSIS
A better testing of the
check if the output is (1,
-0.0829, 0.0054), which

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.
- **Autotuning**
  - Critical for performance to provide tuning that is 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.
- **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