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

Stan Tomov (ICL) and Kwai Wong (JICS)
University of Tennessee

9th JLESC Workshop

April 16, 2019
Acknowledgements:

- Support from NSF, UTK, JICS, NICS, ICL
- Efforts from many colleagues, collaborators, and students
- This project was sponsored by the National Science Foundation through Research Experience for Undergraduates (REU) award no. 1659502, with additional support from the National Institute of Computational Sciences at University of Tennessee Knoxville. This project used allocations from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575. In addition, the computing work was also performed on technical workstations donated by the BP High Performance Computing Team.
- [www.jics.utk.edu](http://www.jics.utk.edu), [icl.cs.utk.edu](http://icl.cs.utk.edu), [cfdlab.utk.edu](http://cfdlab.utk.edu), [www.jics.utk.edu/recsem-reu](http://www.jics.utk.edu/recsem-reu)
Data Driven + Compute Intensive Modeling: openDIEL + MagmaDNN

MagmaDNN: Data Driven Computing, MPI, C++, CUDA GPU, Mixed Precision

openDIEL: Code Based Systems Simulations, MPI, C, Workflow, I/O

- A unified workflow framework to facilitate system-wide inter-disciplinary simulations for data driven and compute intensive modelings on HPC platforms

![Workflow Diagram]

Load Data (cvs, images)

Preprocessing shape, store in tensors

Create/Load Model set hyperparameters

Train Model

Export Model, Predict
MagmaDNN 0.2

High-Performance Data Analytics for Manycore GPUs and CPUs
Written in C++ with clear and reusable interfaces

https://bitbucket.org/icl/magmadnn

Undergraduate Students:
Lucien Ng¹, Sihan Chen¹, Alex Gessinger³, Daniel Nichols², Frank Betancourt²
¹ The Chinese University of Hong Kong
² The University of Tennessee, Knoxville (UTK)
³ Slippery Rock University

Kwai Wong¹,², Stanimire Tomov³, Azzam Haidar⁴, Ed D’Azevedo², Jack Dongarra³,²
¹ The Joint Institute for Computational Sciences (JICS), UTK
² Oak Ridge National Laboratory (ORNL)
³ The Innovative Computing Laboratory, UTK
⁴ Nvidia Corporation

Research Experiences for Computational Sciences, Engineering, and Mathematics (RECSEM): 2013-2019

Direct a group of undergraduate students to explore the emergent interdisciplinary computational science models and techniques in compute and data intensive applications on HPC platforms.

https://bitbucket.org/icl/magmadnn
Data Analytics and LA on many small matrices

Accelerating CNNs with Winograd’s minimal filtering algorithm on GPU

- A machine learning framework built around the Magma BLAS aimed at providing a modularized and efficient tool for training deep nets.
- MDNN makes use of the highly optimized Magma BLAS giving significant speed.
- Memory Management, Mixed Precision, Multiple GPUs, CNN, Grid search engine
- Distributed, Imagenet Test, RNN, GAN, GUI, Hyperparameter optimization .. Coming this summer

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.

Open source; looking for feedback and contributions

Started with students from REU/RECSEM program

https://bitbucket.org/icl/magmadnn
Organization of MagmaDNN

**Initialized/Finalize:** Since MagmaDNN uses Magma we need to initialize/finalize the magma library at the beginning and end of our code.

```cpp
#include "magmadnn.h"
using namespace MagmaDNN;

magma_init();
magma_print_environment(); // optional
... code body here ...
magma_finalize();
```

MDNN uses tensors as its core data structure to store multidimensional data.

```cpp
// Tensors are in the MagmaDNN namespace
Tensor<float> example_tensor {{100, 10, 5}};
```

A Model is comprised of two elements:
1. A vector of NN layers beginning and ending with input and output layers.
2. Hyperparameters: learning rate, weight decay, batch size, and epochs.

All layers require a specified precision (float or double) and a pointer to the previous layer. The available layers are *Input*, *FC*, *Activation*, *Conv2D*, *Pooling2D*, *Flatten*, and *Output*.

```cpp
InputLayer<float> input_layer (x_batch_tensor);
FCLayer<float>* fully_connected_layer_1 = new FCLayer<float> (&input_layer, 512);
ActivationLayer<float>* activation_layer_1 = new ActivationLayer<float> (fully_connected_layer_1, SIGMOID);
FCLayer<float>* fully_connected_layer_2 = new FCLayer<float> (activation_layer_1, 10);
ActivationLayer<float>* activation_layer_2 = new ActivationLayer<float> (fully_connected_layer_2, RELU);
OutputLayer<float>* output_layer = new OutputLayer<float> (activation_layer_2, y_batch_tensor, BIN_CROSSENTROPY_WITH_...
```

```
std::vector<Tensor<float>> layers = {[&input_layer, fully_connected_layer_1, activation_layer_1, fully_connected_layer_2, activation_1...
```

```cpp
// x is the sample tensor
// ret (tensor) is assigned the probability for each output class
dnn_model.predict(x, ret);
```
- Allow various machine learning frameworks and compute intensive applications to run concurrently
- Launch multiple copies of same program seamlessly in different I/O directories
- Communicate data among programs via tuple space IEL function call
- Allow linear and DAG dependency for groups and sets of modules (program)
- mpirun --np xx./openDIEL-AM workflow.cfg
- -- dynamic task scheduling, distributed DNN optimization, in-situ I/O, applications needed
High Performance Traffic Assignment Based on Variational Inequality

Students: XIAO Yujie (CityU), SHI Zhenmei (HKUST)
Mentors: Cheng Liu (ORNL), Kwai Wong (UTK)

Abstract

Variational Inequality (VI) is a mathematical problem that is widely applied to equilibrium problems in different fields. This project focuses on modeling the transportation assignment problem using variational inequality and solving it in a highly efficient way. The solver implements on GPU using parallel computing. The problem can be classified into two categories Static Traffic Assignment (STA) and Dynamic Traffic Assignment (DTA). STA, in this project, refers to solving for the user equilibrium which is a Nash equilibrium about the travel cost of each user without considering about demand changing over time. DTA refers to solving for the dynamic user equilibrium where the cost of travel is minimized for every user, in a continuous period of time.

Static Traffic Assignment

Nonlinear Complementarity Problem (NCP)

Given a mapping $F: R^n \rightarrow R^n$, the NCP $(F)$ is to find a vector $x \in R^n$ satisfying $0 \leq x \perp F(x) \geq 0$.

Static traffic assignment (STA):
The STA problem in the NCP formulation as following:

\[ 0 \leq C_p(h) - u_{wp} \perp b_p \geq 0, \quad \forall w \in \mathcal{W} \text{ and } p \in \mathcal{P}_w; \]

\[ \sum_{p \in \mathcal{P}_w} b_p = d_o(u), \quad \forall w \in \mathcal{W}, \]

\[ w \in \mathcal{W}. \]

\[ F(h, u) \equiv \begin{pmatrix} C(h) - \Omega^T u \\ \Omega h - d(u) \end{pmatrix}, \]

In this formulation, find the candidate paths between each Origin – Destination (OD) pair is important.

Algorithm and Implementation

Step 1 Use shortest path algorithm to find 7 paths for each OD pair. Here the solver uses nvGRAPH package in CUDA library which runs on GPU.

Step 2 Convert all data into NCP formulation.

Step 3 Use NCP FBLSA Algorithm to solve the problem with given error bound. Here the solver uses Siconos package which is non-smooth numerical simulation and cuSPARSE library which also runs on GPU.

Dynamic Traffic Assignment

Dynamic traffic assignment (DTA) models determine departure rates, departure times and route choices over a given planning horizon.

Dynamic network loading (DNL) is a critical procedure for obtaining the traffic delay operator, which maps a traversal time to the departure time. In this project, the DNL procedure is approximated by a system of ODEs:

\[ \frac{dC_p}{dt} = C_{p_0} - C_{p} \quad \forall p \in \mathcal{P}, \quad i \in [1, m(p)] \]

\[ \frac{dx_p}{dt} = x_{p_0} - x_p \quad \forall p \in \mathcal{P}, \quad i \in [1, m(p)] \]

\[ h_{p}^{t_t}(t) = g_{p_0}(t + D_x(x_{p_0}) \{ (1 + D_{x_{p_0}} x_{p_0}) \}) \]

\[ g_{p_0}(t) = g_{p_0}(t + D_{x_{p_0}} x_{p_0}) \{ (1 + D_{x_{p_0}} x_{p_0}) \} \quad \forall p \in \mathcal{P}, \quad i \in [2, m(p)] \]

Differential variational inequality (DVI) is a particular branch of variational inequality (VI). And we employ it to formulate the DTA problem.

Fixed point iteration:
The original DVI problem is restated as a fixed point iteration in the computing process as following:

\[ h^{t_{t+1}}(t) = h(t) - \psi_{p_0}(h(t)) + r_{t+1} \quad \forall t \in \mathcal{W}, \quad \forall p \in \mathcal{P}_w \]

\[ \sum_{t \in \mathcal{W}} h^{t_{t+1}}(t) - \psi_{p_0}(h(t)) + r_{t+1} = Q \quad \forall t \in \mathcal{W} \]

Algorithm

For each OD pair, given $h_0$

\[ \sum_{i} \| h^{t_{t-1} - h_i} \| / \| h_i \| = \varepsilon \]

\[ ODE(h^{t_{t+1}}) = makeODE(h^{t_{t+1}}) \]

\[ x = sol. \text{ to } ODE(h^{t_{t+1}}) \]

For $i = 1 \text{ to } m$

\[ \xi_i = t + D_x(x(\xi_i)) \]

\[ \xi_{i+1} = \xi_i + D_x(x(\xi_i)) \]

End for

\[ D_0 = \xi_m \]

Cost = $\psi(h)$

Example and Results

For the network as shown in the upper left graph, the assignment result can be shown on the graph. The running time can be reduced to one third compared with Frank-Wolfe algorithm in this example. GPU acceleration is significant.

Future Work

For STA, the parallel computing is simply link CUDA library. Optimization in parallel method for this specific problem is still available.

For DTA, the work already done is still in the early stage, there remains a lot of work to optimize the algorithm, including verifying the convergence of the algorithm theoretically.

Acknowledgements

Support from National Science Foundation, Joint Institute for Computational Sciences, University of Tennessee, Hong Kong University of Science and Technology, City University of Hong Kong, and Oak Ridge National Laboratory are gratefully acknowledged.
Actual Traffic in a Virtual Reality
Michael Han,
Mentors: Cheng Liu, Kwai. Wong

Abstract
The purpose of this project is to integrate real-time traffic surveillance footage into a virtual reality setting. This can not only help with better projecting where any given car could go, but can also potentially improve driving simulations by giving their 3D environments some real-world traffic, making them more realistic. We are hoping to achieve this goal using primarily OpenCV/OpenGL and Unity.

Current Results
As of yet, step 1 is done. The method of analysis we ended up writing for this project involved OpenCV and OpenGL.

By blurring two frames within a very short period of time, we can find the changes in light between the both instantaneous pictures, allowing us to detect the moving vehicles. The result is usually faint, grey smudges on a black screen, so using dilation and threshold filters brings out the differences between the frames.

We can then conclude that the resulting white shapes are the cars, and then we draw green rectangles over them. Keeping track of the coordinates of the corners of the rectangles allows us to count the cars that have passed either side of the interstate.

What Now?
Now that we have our footage analyzed, we can start integrating 3D into the project. We need to generate shapes representing the cars and trucks.

We have thought of two ways of doing this:

• Generate shapes using OpenGL in the video analysis’s OpenCV code. This would put the cubes directly on the traffic footage.
• Generate shapes in Unity. This would involve feeding the coordinates of the detected cars from the OpenCV code into Unity.

Either way, we would have to make sure that the cubes follow the coordinates generated by the traffic analysis code, and remain the same size (because the size of the green squares fluctuate wildly in our analyzed footage).

Our next goal is to translate the positions of the cubes into a 3D setting, because our traffic footage is in 2D and therefore only showing the traffic from a single perspective. We are thinking about generating a plane/grid representing the road, and writing an algorithm to tilt and rotate the “road” until it matches the lanes in our footage. We can then use the coordinates of our cubes with their relative positions on the plane/grid to be able to generate a 3D environment where one can view the traffic from any perspective.

The Plan
1. Write code that detects vehicles, more importantly, details the exact coordinates where they are on the traffic footage. ✓
2. Write code that generates 3D cubes representing the vehicles and locking on to the their coordinates. □
3. Write code that generates a 3D plane matching the orientation of the road on the traffic footage. □
4. Create the full 3D environment using the objects. □

Acknowledgements
This project is supported by the National Science Foundation.

Thanks to Dr. K. Wong for letting me onboard on this project and thanks to Dr. Cheng Liu for giving me three books on Unity and OpenCV.
Unmixing 4-D Ptychographic Images
Part A: Algorithmic Approach

Students: Michaela Shoffner (UTK), Zhen Zhang (CUHK), and Huanlin Zhou (CUHK)
Mentors: Dr. Richard Archibald (ORNL), Dr. Azzam Haidar (UTK), Dr. Stanimire Tomov (UTK), and Dr. Kwai Wong (UTK)

Abstract

Fast electron detectors are gaining ground in traditional high-resolution microscopy studies. In particular, 4D ptychographic datasets collected over a range of real and reciprocal space coordinates are believed to contain a wealth of information about structure and properties of materials. However, currently available data analysis methods are either too general, only allowing for analysis of simplified objects, or too restrictive, effectively recreating traditional detectors from these datasets before interpretation. This project aims to explore the ways that symmetry model analysis, the tool used to a great effect in theoretical studies of materials, can be adapted to analyze 4D datasets of materials such as multifunctional complex composites.

Goals

Given Matlab code using the least square model unmixing method and three sample base-model data files, we seek to improve upon the baseline program. Our primary two goals are to implement an improvement to the current unmixing algorithm, and to improve speed and performance by converting the program to C code, using LAPACK, and then having it run on a GPU, using MAGMA.

In the case of our problem, it can be formulated as:

\[ aM_1 + bM_2 + cM_3 = b \]

where \( a, b, \) and \( c \) are the coefficients we're trying to find, each \( M_1 \) is a matrix representing one of the baseline models, and \( b \) is the given image. Represented as a grayscale picture, each image looks something like this.

Research Questions

Since our example data has known coefficients, yet the original program does not find particularly close numbers, our task is to seek ways to get greater accuracy in the calculated values.

Starting from a basic least squares algorithm and a basic linear model, we began implementing various methods to solve this unmixing problem.

Process and Algorithms

In the attempt to better solve for the coefficients, we used three different methods, but the workflow for each was still quite similar:

- Matlab Code
- C code for GPU with LAPACK
- C code for GPU with MAGMA
- C Parallelizing

Least Squares

For \( A = [M_1, M_2, M_3] \), \( b \) is the final image data, and \( X = [\alpha, \beta, \gamma] \), the basic least squares is implemented by changing \( A^T A \) into \( A^T A X = A^T b \), as this allows for the overdetermined matrix to be solved easily using LAPACK functions, but since it can be greatly affected by outliers and there is something missing in the simple linear model, it is also the least accurate method.

Simplified Least Squares with gradient

4 representative biases:
\( \alpha M_1 + \beta M_2 + \gamma M_3 = b \)
where \( \alpha, \beta, \gamma \) are the true weights

Gradients:
(of the 3 modes)

As seen in the figures, both bias and gradient are highly symmetric and have similar patterns, so the gradient might be the missing part. Thus, we design a new model by including the gradients, formulated as:

\[ b = aM_1 + bM_2 + cM_3 + xg_1 + yg_2 + zg_3 + c + dg_0 + e + fg_1 + y + g_0 \]

Split-Bregman Method

This method is used to solve the L1-regularized least squares problem: apart from solving the basic least square form, we also want to minimize the L1-norm of the gradient of the resulting weight matrices, so that the resulting weights are more piecewise constant, just as the true weights appear.

Results

All of our programs are run on the Bridges system.

Both simplified least squares with gradient and the Split-Bregman get greater accuracy than basic least squares, though they are quite similar to each other, with Split-Bregman being slightly better.

The total difference in results from actual values is:

- \( 11.6405 \) for basic least squares
- \( 2.9635 \) for simplified least squares with gradient
- \( 2.8782 \) for the Split-Bregman method

As for speed, both of the latter two methods have very similar speeds and are faster than the equivalent implementation for basic least squares.

Future Work

Next steps include:
- Working with OpenACC to parallelize the code
- Debugging and further streamlining what currently works
- Getting the MPI least squares and GPU simplified least squares plus gradient working

Acknowledgements

With thanks to all of our mentors, who have guided us every step of the way.

This work used the Extrime Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

This project was sponsored by Oak Ridge National Laboratory, the Joint Institute for Computational Sciences, the University of Tennessee, Knoxville, and the Chinese University of Hong Kong.

This project was made possible by support from the National Science Foundation.

Last but not least, thanks to Dr. Google, without whom this would have been impossible.
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 $(0.9950, 0.8929, 0.7545)$. For $(\alpha, \beta, \gamma) = (1, -1, -1)$, the result of least square is $(0.9428, -0.3689, -0.3500)$, which has large relative 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 $(0.9994, -0.9675, -0.9898)$, with 2 hidden layers, 15 nodes in each hidden layer and regularisation parameter = 0.01.

METHOD

Currently, only 16 input images are provided and every four of them share the same set of coefficients. Namely,

$$(\alpha, \beta, \gamma) = (1, \pm 1, \pm 1)$$

This shortage of data makes it impossible to train a neural network with what we now have. The remedy is to 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 $\beta$ and $\gamma$. 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.

The activation function is tanh except for output layer, which is the identity.

Different interpolation methods can be chosen. The figure on the right shows the 6-point cubic interpolation of one pixel.

The cost function is minimised by a CG method. With 200 training examples, regularisation parameter = 0.01, and linear interpolation, the changes of cost functions are shown in the following figure.

Note that in the 4-point case the training is much faster.

RESULTS

<table>
<thead>
<tr>
<th>(4-point case, one input image for each set of coefficients)</th>
<th>True coef</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.9891</td>
<td>1.0033</td>
<td>0.9823</td>
<td>0.9925</td>
<td>0.9911</td>
<td>1.0051</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1.0010</td>
<td>0.9735</td>
<td>-0.9755</td>
<td>-1.0396</td>
<td>-0.0494</td>
<td>1.1258</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.9946</td>
<td>-0.9931</td>
<td>0.9914</td>
<td>-1.0123</td>
<td>1.1155</td>
<td>0.0359</td>
</tr>
</tbody>
</table>

(6-point case, one input image for each set of coefficients)

<table>
<thead>
<tr>
<th>True coef</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.9934</td>
<td>1.0019</td>
<td>0.9772</td>
<td>0.9683</td>
<td>0.9947</td>
<td>1.0226</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.8718</td>
<td>1.0750</td>
<td>-1.0476</td>
<td>-0.9736</td>
<td>0.9907</td>
<td>0.0199</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1.0464</td>
<td>-0.9962</td>
<td>0.9614</td>
<td>-0.9915</td>
<td>0.0310</td>
<td>1.0205</td>
</tr>
</tbody>
</table>

ANALYSIS & FUTURE WORK

A better testing of the algorithm is to directly input $M_0$ and check if the output is $(1,0,0)$. In fact, the output is $(1.0050, -0.0829, 0.0054)$, which is quite close. This successful prediction on $M_0$ gives us confidence that this interpolation-training method can work for other test data.

In the future, if more data can be experimentally acquired, this algorithm can be more rigorously tested. If it does not work well with the new data, we can interpolate with the new data and thus improve the model.

Implementation of the algorithm using MAGMA is being worked on.

Essentially, this neural network is solving this problem similar to using least square method, except that there is a non-linear bias of unknown mathematical form. Therefore this idea can possibly be used to solve other linear algebra problems based on the NN formulation, such as to compute the matrix inverse.

ACKNOWLEDGEMENTS

This program is sponsored by the National Science Foundation, the Joint Institute of Computational Sciences, and the Chinese University of Hong Kong.

REFERENCE

1. imc@q, Carl Edward Rasmussen, 2003