Acceleration of multivariate analysis techniques in TMVA using GPUs

To cite this article: A Hoecker et al 2012 J. Phys.: Conf. Ser. 396 022055

View the article online for updates and enhancements.

Related content
- Algorithm Acceleration from GPGPUs for the ATLAS Upgrade
  P J Clark, C Jones, D Emelyanov et al.
- The toolkit for multivariate data analysis, TMVA 4
  P Speckmayer, A Höcker, J Stelzer et al.
- Ideal tagging with the multivariate data-analysis toolkit TMVA
  A Heikkinen, P Kaitaniemi, V Karimäki et al.

Recent citations
- Many-core on the Grid: From exploration to production
  T Doherty et al
Acceleration of multivariate analysis techniques in TMVA using GPUs

A. Hoecker\textsuperscript{1}, H. McKendrick\textsuperscript{2}, J. Therhaag\textsuperscript{3}, A. Washbrook\textsuperscript{2}

\textsuperscript{1} CERN, Switzerland
\textsuperscript{2} School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom
\textsuperscript{3} Univ. Bonn, Physikalisches Inst. Nussallee 12, D-53115 Bonn, Germany

E-mail: awashbro@ph.ed.ac.uk

Abstract. A feasibility study into the acceleration of multivariate analysis techniques using Graphics Processing Units (GPUs) will be presented. The MLP-based Artificial Neural Network method contained in the TMVA framework has been chosen as a focus for investigation. It was found that the network training time on a GPU was lower than for CPU execution as the complexity of the network was increased. In addition, multiple neural networks can be trained simultaneously on a GPU within the same time taken for single network training on a CPU. This could be potentially leveraged to provide a qualitative performance gain in data classification.

1. Introduction

Multivariate classification methods based on machine learning techniques are commonly used for data analysis at the LHC in order to look for signatures of new physics beyond the standard model. A large variety of these techniques are contained in the Toolkit for Multivariate Analysis (TMVA) which enables training, testing, performance evaluation and application of the chosen classification methods.

As data continues to be successfully collected at the LHC at record rates it is foreseen that the sample size processed by TMVA is expected to grow by orders of magnitude. However, it is known that some classification techniques are likely to be processor bound as the sample size is significantly increased. Other input factors, such as the number of classifier variables defined for a given method, can also lead to an appreciable increase in overall execution time. Investigating opportunities for potential performance improvements is therefore of great importance.

2. Toolkit for Multivariate Analysis

The Toolkit for Multivariate Analysis (TMVA) provides a ROOT-integrated \cite{1} environment for the application of supervised learning techniques used to separate events which can be classified as scientifically significant signal from those considered to be background. These techniques include:

- Rectangular cut optimisation
- Projective likelihood estimation
- Multi-dimensional likelihood estimation
• Linear and nonlinear discriminant analysis
• Artificial neural networks
• Support vector machine
• Boosted/bagged decision trees

Each of the above techniques have specialisations that are enabled by user input parameters. A full description of the TMVA framework and the classification techniques it supports can be found in [2].

The overall workflow of TMVA is shown in Figure 1. Each multivariate technique makes use of input training events, for which the desired output is known, to determine a mapping function describing a decision boundary. Testing events are then used to evaluate classification performance. A transparent factory pattern allows TMVA to remain agnostic about which of its methods are being used for analysis as methods are created and handled in terms of their abstract base class. This ensures that each method can be applied to the training and test data sets in the same way, ensuring results are not biased by the framework.

![Figure 1. Workflow of a typical TMVA training application. TMVA consecutively calls the training, testing and performance evaluation methods for each selected classification technique.](image)

2.1. Classification Performance
The time taken to train a selection of classification techniques in TMVA is shown in Figure 2. Linear-based classifiers such as cut optimisation and likelihood estimation are in general much faster to train than non-linear methods. The difference in training times is also more apparent using a larger input sample. This is not only due to the larger number of events but also to the higher number of input variables in the sample. This increase in parameter space can cause a sharp rise in processing time for some classification methods.
Figure 2. Training times for a selection of TMVA classification techniques. Timing values are shown for both the input data samples used for GPU evaluation (Section 4.2)

Note that the time taken for training is not the primary indicator of classifier performance. For example, a cut based optimisation may be faster to train but may not give the best discriminating power. It is therefore up to the user to determine the appropriate balance between achieving increased discrimination between signal and background and the extra time incurred during training. For increasingly complex analysis problems the extra time needed may exceed what is considered to be reasonable.

In general the time taken for training a classification method is substantially longer than the testing time. As such, the focus of the study was on the potential for reduction of training times. In particular, artificial neural networks were chosen as an area of study based on the relatively high training times for this technique.

2.2. Artificial Neural Networks

Artificial Neural Networks (ANN) use layers of interconnected artificial neurons to model relationships between input and output data. Through iterative training of the network, synapse weights and values can be adjusted to allow input data to be passed between connected neurons and classified appropriately. Feed-forward neural networks (FFNN) are a simple and popular form of an ANN, in which data passes only in one direction between input and output, with no loops or cycles in the network.

Multi-layer perceptrons (MLP) are a sub-class of FFNN formed of a multiple layers of neurons, constructed with one single input layer, one or more “hidden” layers and single output neuron. Figure 3 outlines the structure of a basic MLP network.

Input data is sequentially fed through the network using each of its event variables as input to a separate neuron on the input layer. This layer is then connected by a number of weighted synapses to a hidden layer, which can then be connected in turn to additional hidden layers. Individual neurons in an MLP network accept a number of weighted inputs through their synapses to form a single output value calculated using activation threshold and neuron response functions.

Finally, the data exits the network through the output neuron to provide a single classification result. Since the desired output for training data is known, this result can be used to adjust synapse weights by the application of a back-propagation algorithm. These steps are repeated using the training data for a number of iterations (or “epochs”) to optimise the classification performance. More details on the activation, response and back-propagation functions used in
the MLP implementation in TMVA are found in \cite{2}.

3. General Purpose GPU computing
The need to seek performance gains through increased parallelism rather than through processor improvements has been apparent for a number of years. Alternative highly-parallel hardware architectures, such as GPUs, have led to performance gains in a number of disciplines. This is now being adopted in High Energy Physics especially for time-critical computing environments such as the ATLAS Trigger \cite{3}.

GPUs are optimised for high-throughput data-parallel processing of floating point numbers and the computational benefits stem from dedicating more transistors on a GPU to data processing at the cost of flow-control and caching available on a CPU. Careful consideration must therefore be given to memory allocation, host to device transfer timing and the effect of code branching across multiple threads of execution.

A number of platforms exist for GPU code development, such as CUDA \cite{4} and OpenCL \cite{5}, which allow code development in high level languages (C, Fortran, C++) and access to a GPU's memory and instruction set. For this study CUDA was selected for this project due to its current popularity and the quantity of resources available.

In the CUDA model, the GPU acts as a co-processor with overall program flow control maintained by the CPU and data transfers and allocation initiated from the host. Parallel operations are organised into a hierarchy of threads providing a logical way to address data structures such as vectors, matrices and arrays of one or more dimensions. A *kernel* represents a single module of computation that can be performed in parallel by a number of threads.

*Thread blocks* consist of single or multi-dimensional arrays of threads which can be coordinated using synchronisation primitives to ensure every thread reaches a certain point before computation can continue. Threads within a block will reside on the same physical multi-processor core on the GPU which constrains the number of threads per block to the underlying GPU architecture. No guarantee can be made about execution order and so each thread block must be able to be processed independently. This model allows simultaneous thread block execution to scale across multiple GPUs.

4. Development and Performance Testing
There are a number of areas in TMVA which could benefit from the parallelism available from GPUs. As discussed in Section 2.1, it was decided to isolate one chosen classification technique...
and modify the implementation of the algorithm for GPU execution. This enabled a realistic evaluation of the benefits of GPUs without the need for wholesale structural changes to the TMVA framework.

4.1. Parallelism approach
The existing MLP routine was profiled to determine the methods of parallelisation available. It was found that event-based parallelism could not be considered due to the implicit training dependency from prior input data. Synapse weights in a neural network are iteratively adjusted by successive training input data and so events could not be processed in parallel. However, parallelism was achieved by creating a set of kernels to calculate the activation, response and back propagation functions simultaneously for each neuron in the network. Synchronisation barriers were placed between each of the neuron calculation steps to avoid race conditions.

4.2. Performance Evaluation
The evaluation of the GPU-based MLP routine was made by using two hosts containing different models of GPUs. The characteristics of the GPUs used for testing are shown in Table 1. The table also includes the details of the CPU on each host which needs to be considered for any interpretation of timing performance.

<table>
<thead>
<tr>
<th>Setup</th>
<th>GPU model</th>
<th>MP</th>
<th>Cores</th>
<th>Global Memory</th>
<th>Shared Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nvidia C1060 (Tesla)</td>
<td>30</td>
<td>240</td>
<td>4096 MB</td>
<td>16 KB</td>
</tr>
<tr>
<td>2</td>
<td>Nvidia C2050 (Fermi)</td>
<td>14</td>
<td>448</td>
<td>2687 MB</td>
<td>48 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Setup</th>
<th>CPU model</th>
<th>Frequency</th>
<th>Cache Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intel Xeon X5560</td>
<td>2.8 GHz</td>
<td>8192 KB</td>
</tr>
<tr>
<td>2</td>
<td>Intel Xeon E5502</td>
<td>1.9 GHz</td>
<td>4096 KB</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of the GPU (upper table) and CPU (lower table) models used for performance evaluation. MP is the number of multi-processors resident on the GPU.

Two input data samples were used for training time comparison (shown in Table 2). The “small” sample was a reference sample packaged with TMVA. The “large” sample was a Monte Carlo simulation dataset used in the analysis of data from the ATLAS experiment and contained a much higher number of events, and a higher number of variables per event. Each data sample consisted of known signal and background events that could be used to train a neural network. The distributions for an selection of input variables from the large sample are shown in Figure 4.

<table>
<thead>
<tr>
<th>Sample name</th>
<th>Input Variables</th>
<th>Number of events</th>
<th>Neurons</th>
<th>Synapses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>4</td>
<td>6000</td>
<td>15</td>
<td>49</td>
</tr>
<tr>
<td>Large</td>
<td>35</td>
<td>32486</td>
<td>77</td>
<td>1444</td>
</tr>
</tbody>
</table>

Table 2. Input data samples used for performance evaluation. The number of neurons and synapses is derived from the default MLP network configuration in TMVA. The network consists of one hidden layer containing $N+5$ neurons, where $N$ is the number of input variables.

5. Results
The timing results for both the CPU and GPU-based MLP methods are shown in Table 3. It is seen that the GPU-based version provides execution speed-up but only for the large sample. This
Figure 4. A selection of input variable distributions taken from the large data sample. It can be seen that some variables exhibit a higher degree of separation between signal and background.

Inconsistency is explained by considering the time taken for kernel initialisation and the overall utilisation of the GPU. For the small sample test, a non-negligible fraction of execution time is taken to transfer input data across to the GPU and to initialise kernel operations. Furthermore, only a small proportion of the GPU is actively used for parallel processing. A relatively small number of input variables per input event results in a lower amount of neurons in a network and so only a small degree of parallelisation can be obtained.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Sample name</th>
<th>CPU time (sec)</th>
<th>CPU + GPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Small</td>
<td>19</td>
<td>121</td>
</tr>
<tr>
<td>1</td>
<td>Large</td>
<td>930</td>
<td>667</td>
</tr>
<tr>
<td>2</td>
<td>Small</td>
<td>34</td>
<td>223</td>
</tr>
<tr>
<td>2</td>
<td>Large</td>
<td>1830</td>
<td>1180</td>
</tr>
</tbody>
</table>

Table 3. Training times for the CPU and GPU based versions of the MLP classification method.

In contrast, the large sample has both a longer operational time (due to the data sample size) and a higher number of variables per event. The proportion of running time spent in kernel and data initialisation steps is therefore reduced and a larger degree of parallelisation available which is reflected in the shorter execution time.

Performance improvements were not expected with an increase in data sample size due to the dependency on prior events during network training. This was illustrated by creating a new series of input data of increasing size derived from the large input sample (Figure 5a). Here, the training time for the GPU-based MLP was observed to scale with data sample size without any appreciable reduction. Similarly, training time was also expected to scale with the number of training epochs. In the large sample case where the GPU-based version of MLP is faster (Figure 5b) the difference in training times will start to increase as more training iterations are performed.
5.1. Network Complexity
In general it was observed that the performance benefits from GPU execution were dependent on the complexity of the underlying network being trained. The complexity can be increased by adding additional hidden layers to a network.

Both the standard and GPU-based MLP routines were tested with up to five additional layers in each network. The number of neurons in each additional layer was set to $N+5$ and $N+10$, where $N$ is the number of neurons in the input layer. The amount of neurons and synapses in these networks are shown in Table 4 and the effect of changing the network on training time is shown in Figure 6a.

For the CPU-based routine there was an expected scaled increase in training time with the number of additional layers in both the $N+5$ and $N+10$ cases. However, it can be seen that the training time remained constant regardless of the number of layers chosen and in the number of neurons in each additional layer. The training time was expected is to eventually increase as the GPU became fully utilised.

5.2. Training of multiple networks
The ability to perform neuron calculations simultaneously implied that it was also possible to train multiple independent neural networks in parallel on the same GPU. Figure 6b shows the training time for both data samples for an increasing amount of networks being trained. It can be seen that training times remain constant and that up to 128 (256) networks can be processed simultaneously for the large (small) sample before GPU memory limitations

Figure 5. Training time performance for (a) an increasing number of input events, (b) an increasing number of training epochs.
Figure 6. Training time performance for (a) an increasing number of hidden layers in a MLP network, (b) an increasing number of independently trained MLP networks.

are reached. Memory exhaustion can be remedied by greater use of shared memory rather than global memory on the GPU. This would allow scalability beyond one GPU device and theoretically allow any amount of networks to be processed in parallel.

Table 4. Number of neurons and synapses contained in a MLP network with an increase in the number of hidden layers.

<table>
<thead>
<tr>
<th>Layers</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neurons  ((N + 5))</td>
<td>15</td>
<td>24</td>
<td>33</td>
<td>42</td>
<td>51</td>
<td>60</td>
</tr>
<tr>
<td>Synapses ((N + 5))</td>
<td>49</td>
<td>121</td>
<td>193</td>
<td>265</td>
<td>337</td>
<td>359</td>
</tr>
<tr>
<td>Neurons  ((N + 10))</td>
<td>20</td>
<td>34</td>
<td>48</td>
<td>62</td>
<td>76</td>
<td>90</td>
</tr>
<tr>
<td>Synapses ((N + 10))</td>
<td>79</td>
<td>261</td>
<td>443</td>
<td>623</td>
<td>807</td>
<td>989</td>
</tr>
</tbody>
</table>

This form of parallelism has two applications for classification training. The TVMA framework could be modified to run in batch processing mode on a GPU cluster to handle asynchronous network training requests. However, the motivation for such a use case is low in high energy physics and currently does not match user requirements.

Alternatively, the performance of the neural network technique could be improved by determining the best network structure for a given classification problem. Multiple networks could be constructed with a range of input parameters and trained in parallel to evaluate the
network that provides the best performance. In addition, the optimum network could be derived programmatically by using a gradient-descent method to determine the best result.

An example of this approach is shown in Figure 7. A cut value on the MLP network output is used to reject the background sample whilst retaining as much signal sample as possible, and is quantified by the significance value (Figure 7a). The variation and magnitude in significance will be different for each network and may yield better results in some cases.

Figure 7. (a) Signal and background cut efficiencies and significance for a sample MLP network. (b) Background rejection versus signal efficiency (ROC curve) for multiple MLP networks. ROC curve tending to the top right corner of the plot indicate better performance.

A comparison of background rejection versus signal efficiency for several networks is shown in Figure 7b. An ideal MLP network would allow all background to be rejected whilst keeping close to 100% signal efficiency. It can be seen there can be variation in network performance between different networks. Although network optimisation appears to be feasible from this study, some care would be needed to ensure a large enough training sample is used to avoid overtraining.

6. Conclusions

A feasibility study into the acceleration of MLP-based artificial neural network contained in TMVA has shown that a large degree of parallelisation is possible by the use of the GPUs. A neuron-based parallelism development approach allowed neural networks to be processed faster as the complexity of the chosen network was increased. Furthermore, multiple networks could be processed in parallel potentially leading to a qualitative performance gain in classification.

Although initial results were encouraging, a full validation of classification results will be needed before this can be used as a viable alternative to the current MLP technique. A notable omission in the GPU-based MLP code was the use of a bias term [2]. This is an additional neuron added to each of the non-output layers and enables faster convergence of the network. This feature was not implemented due to code branching concerns in the implemented kernels but will need to be included in the GPU-based method to provide equivalent results.

In addition, there is scope for using the memory on a GPU more efficiently. The speed of data access during execution could be improved by packing data into shared memory used in a thread block and would allow for scalability beyond a single GPU device. The use of caches on newer GPU devices may also yield better timing performance.

The GPU ecosystem is a rapidly advancing and so any future effort would have to be mindful of future changes to GPU device architectures as this will directly impact on development strategy. For example, the recent Nvidia Kepler GPU series now supports a form of dynamic
parallelism [6] which could have potential benefits for this study, especially in the area of network parameter optimisation (Section 5.2).

At this juncture, a decision is needed on whether to make significant changes to TMVA code to arrange input data more effectively for GPU execution or if separate GPU-based versions of existing classification techniques is a more preferable approach. Although the study has focused on just one of several techniques most of the practices exercised in this study can be applied across the TMVA framework.

References