Acceleration of ensemble machine learning methods using many-core devices

A. Tamerus\textsuperscript{1}, A. Washbrook\textsuperscript{1}, D. Wyeth\textsuperscript{2}

\textsuperscript{1} High Performance Computing Service, The University of Cambridge, Roger Needham Building, 7 JJ Thomson Avenue, Cambridge CB3 0RB, United Kingdom
\textsuperscript{2} SUPA, School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom

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

Abstract. We present a case study into the acceleration of ensemble machine learning methods using many-core devices in collaboration with Toshiba Medical Visualisation Systems Europe (TMVSE). The adoption of GPUs to execute a key algorithm in the classification of medical image data was shown to significantly reduce overall processing time. Using a representative dataset and pre-trained decision trees as input we will demonstrate how the decision forest classification method can be mapped onto the GPU data processing model. It was found that a GPU-based version of the decision forest method resulted in over 138 times speed-up over a single-threaded CPU implementation with further improvements possible. The same GPU-based software was then directly applied to a suitably formed dataset to benefit supervised learning techniques applied in High Energy Physics (HEP) with similar improvements in performance.

1. Introduction

Multivariate training and classification methods using machine learning techniques are commonly applied in data analysis at HEP experiments. Despite their success in aiding the search for signatures of new physics beyond the standard model it is known that some of these techniques are computationally bound as input sample size and model complexity are increased. Investigating opportunities for potential performance improvements is therefore of great importance if these techniques are to be applied to much larger data volumes expected from future operations at the Large Hadron Collider.

A case study is presented into the acceleration of decision forests using many-core devices in collaboration with Toshiba Medical Visualisation Systems Europe (TMVSE). The main steps undertaken to convert a decision forest classification algorithm used in medical imaging applications to run on a GPU platform will be discussed. We will outline the development approach and optimisation steps taken to maximise the performance of the GPU-based algorithm and demonstrate how this context independent solution can be applied to HEP data analysis.

2. Medical Context

TMVSE develop software to process three-dimensional medical imaging data (such as CT or MRI scans), using automatic detection of anatomical landmarks defined on the skeleton, vasculature...
and major organs. Landmark detection underpins a semantic understanding of the medical data and thus has many diverse applications, for example, it facilitates rapid navigation to a named organ thereby assisting radiologists performing post-scan analysis.

Supervised ensemble machine learning methods are applied to imaging data to determine whether a given image voxel (a volumetric pixel) can be classified as belonging to one of hundreds of registered landmark sites marked by anatomists. Decision forests [1] are favoured over similar supervised learning methods (such as Support Vector Machines) due to their straight-forward application to multi-class problems and classification transparency.

![Figure 1.](image)

*Figure 1. (Left) Representation of a decision tree algorithm applied to medical images (Right) Visual illustration of automatically detected landmarks positioned on novel CT sagittal scan*

In a recent implementation of this approach [2], decision forests composed of 80 trees were each trained with 40 (out of a possible 369) reference datasets each combining training voxels in the neighbourhood of a landmark with randomly sampled background voxels. Every voxel in a new image was passed down each tree in the forest to calculate a set of normalised likelihoods. For each landmark, the voxel with the greatest normalised likelihood for that landmark was selected as the potential detection point. CT volume data are downsampled for scale independence and rotated to enable consistent orientation across images. The final position was selected by interpolating from the neighbourhood of the sampled voxel with the maximum likelihood for the class. An illustration of the application of a sample decision tree to medical imaging data is shown in Figure 1.

### 3. GPU-based Image Classification

The machine learning implementation described in Section 2 has been optimised to run efficiently on multiple CPU cores to achieve low execution times. After data preparation and optimisation the execution time was on average 4.5 seconds per volume with a sub-second processing time being desirable. A limiting factor in achieving higher performance was found in hardware and cost constraints: TMVSE’s software is intended to run on medical workstations with limited computing resources.

TMVSE are therefore pursuing the investigation of affordable supplemental sources of processing power such as GPUs in order to bring down algorithm execution time. The main advantage of GPUs is their massively parallel nature, processing a large amount of data simultaneously over thousands of multiple threads in a SIMD fashion.

A GPU-based decision tree traversal classification algorithm was developed to determine whether many-core devices could offer a significant performance advantage for image processing.
The development focus was on the image classification process which was considered to be more performance critical. Although more computationally expensive, the training step is performed in advance of the user application and run far less frequently.

The classification process is composed of two distinct steps. The pre-trained random forest is first applied to the volume data in order to obtain a classification result for each tree. All the results are then retrieved and combined to generate a final classification result. The focus of our study was on the first of these steps. Here the computationally intensive tree traversal operation is applied to each voxel independently and so could be more easily mapped to parallel processing on a many-core architecture.

3.1. Testing and Validation Approach
Testing and validation was possible by using an anonymised input data volume and pre-trained decision tree provided by TMVSE and decoupled from their production framework. The volume data described a three-dimensional, voxel-based representation of a CT scan stored in a binary format with dimensions of 512 (x) by 512 (y) by 525 (z) voxels. The sample decision tree contained 4217 nodes resolving to 127 classes stored in a XML format that was parsed using the libxml2 library [3] into an in-memory structure.

A relatively simple implementation of the decision forest algorithm was developed in C and CUDA [4]. The CUDA programming model was chosen due to the available hardware supported this model, and its reduced complexity and strong documentation and community support. A CUDA kernel mapped per-voxel tree traversal to a thread block over the x dimension of the image volume. This approach allowed adjustment and scaling of the kernel parameters to allow different volume sizes to be processed. A CPU-based single-threaded reference implementation of the same classification algorithm was developed in-step with the GPU version for performance comparison studies. These implementations were structurally similar with differences only in the key sections offloaded for GPU execution.

All testing was performed on a dedicated server with dual Intel Xeon 5560 CPUs (providing 16 hyperthreaded cores) and an Nvidia Tesla K40 GPU. This device contained 2880 cores - delivering a peak performance of 4.29 Tflops and 1.43 Tflops for single and double precision floating point operations respectively - and 12 GB of globally addressable memory.

Verification of the classification results for both the reference and GPU-based versions were performed by results validation from a single tree test. Figure 2 shows the visualisation of results generated in Paraview [5] from a single tree applied to a slice of the image volume data.

3.2. GPU Optimisation Steps
Optimisations to the code were made iteratively through the development process using guidance from CUDA profiling results. These included the following:

- Restriction of GPU execution to computationally intensive sections of the algorithm
- Maximising device occupancy by aligning the thread block and grid size to match the image volume dimensions
- Defining data structures to access consecutive GPU memory addresses
- Eliminating the use of pointers in data structures unfavourable on a heterogeneous system
- Selective use of constant memory on the GPU for faster per-thread read access to decision trees
- Reducing the host to GPU device transfer latency overhead by efficiently packing data structures
- Use of single precision variables over double precision where identical results could be achieved
Implementing the decision tree in as few objects as possible
• Using CUDA streams to overlap data transfer with classification execution
• Coalescing global memory writes across threads when storing per-voxel classification results
• Reducing per-thread overhead of arithmetic operations (e.g. restricting division where possible)

Any optimisation performed on the GPU code was also applied to the CPU implementation in order to keep performance comparisons representative. However in some cases optimisations made to the GPU code could not be applied the reference CPU implementation, or was found to reduce performance due to differences in the underlying architecture. In these cases, optimisation steps were only applied on the GPU-based version.

It was found that enabling concurrent memory copies and reducing pointer usage when accessing decision trees yielded the most improvement. The use of coalesced memory writes for storing parallelised per-voxel decision tree output and the selective optimisation of arithmetic operations in CUDA kernels also gave notable gains in performance.

In addition, other possible optimisations were identified but were not fully implemented. For example, the concurrent execution of classification kernels was explored but did not yield immediate improvement due to the high thread occupancy already obtained on the GPU. The use of GPU texture memory optimised for spatial locality was a promising approach and could have lessened the impact of non-contiguous memory accesses.

3.3. Performance Results
The timing results for GPU-based, and single and multi-threaded CPU implementations of the decision forest classification technique are shown in Table 1. The time for decision forest traversal and for the entire program execution is shown for both the processing of a single tree and a forest of 80 decision trees. The results demonstrate a clear performance improvement was possible by using a GPU with speed-up observed of 138 times faster over the single-threaded CPU reference implementation.

The tree traversal algorithm in isolation was found to have much higher speed-up which was tempered by the overhead resulting from data transfer and execution on the GPU. Therefore
timing improvements were more pronounced for a larger amount of trees due to the lower proportional cost of host to device transfer activities. A comparison with a multi-threaded version of the CPU reference implementation (using OpenMP 3.1) with 8 cores showed the GPU version was 19 times faster for a forest of 80 trees. Here, the OpenMP-based version of the algorithm performed well but was limited to linear scaling, and sub-linear scaling for 16 threads where hyper-threaded cores were used.

Timing performance compared favourably to processing times found in the production environment when compensating for pre-processing optimisation steps and downsampling of the input data not applied in this feasibility study. In particular, a reduction in image dimensions would allow the smaller but much lower latency GPU shared memory to be used to store image data for each block of threads in favour of the slower larger capacity global memory used in the implementation.

| Threads | Traversal time (s) | Traversal Speed-up | Execution time (s) | Speed-up |
|---------|-------------------|-------------------|-------------------|----------|
| 1 tree  |                   |                   |                   |          |
| 1       | 18.68             | 1                 | 18.84             | 1        |
| 8       | 2.58              | 7.3               | 2.71              | 7.0      |
| 16      | 1.41              | 13.3              | 1.54              | 12.2     |
| GPU     | 0.10              | 186.3             | 2.67              | 7.1      |

80 trees

| Threads | Traversal time (s) | Traversal Speed-up | Execution time (s) | Speed-up |
|---------|-------------------|-------------------|-------------------|----------|
| 1       | 1466.82           | 1                 | 1466.96           | 1        |
| 8       | 200.68            | 7.3               | 200.80            | 7.3      |
| 16      | 119.29            | 12.3              | 119.41            | 12.3     |
| GPU     | 8.03              | 182.6             | 10.61             | 138.3    |

Table 1. Performance timing results for GPU-based, and single and multi-threaded CPU implementations of the decision forest classification technique. The table lists timing values in seconds for a forest containing a single tree (upper section) and 80 trees (lower section)

4. HEP Applications
The GPU-based decision forest classification technique developed for the processing of medical images was directly applied to the classification of HEP collision event data without loss of functionality. As the GPU code was developed to be agnostic on the content of input data and decision forest composition it meant that minimal changes were required if new input could match the format used in the original application.

4.1. Development Approach
A publicly available dataset created for the ATLAS Higgs Boson Machine Learning Kaggle challenge [6] was chosen as a simple representative example of HEP detector collision data. This choice avoided unnecessary overhead from experiment-specific formats which would have made data conversion less trivial. The main distinction between the two applications was the classification of events into only two classes (signal and background) for the HEP dataset compared with 127 classes in the TMVSE example.

The dataset was composed of 30 feature parameters and a training and testing sample size of 250,000 and 550,000 events respectively. The Scikit learn toolkit [7] was used to construct and train a decision forest composed of a 100 trees based on the HEP dataset. The size of an individual tree was limited to 1000 nodes to compare with the tree composition provided in the medical imaging application.
After the classification step was validated the timing performance was studied by replicating the original testing sample a number of times to determine any scaling limitations with sample size.

4.2. Performance Results

The GPU-based classification timing performance for the processing of the HEP dataset is shown in Figure 3. Overall processing time was observed to be near-independent of sample size up to 55 million events (100 times the original sample size) before the threshold on GPU global memory capacity was reached. The figure illustrates similar processing time up to this threshold compared to the super-linear scaling observed in the CPU-based reference version processing the same dataset.

The timing breakdown of the GPU overhead derived from CUDA profiling is shown for increasing sample sizes in Figure 4. Tree traversal processing is seen to be mostly independent of sample size but small increases in processing time were observed with the transfer of larger datasets.

Slight timing performance improvements were also observed with tree complexity either by replicating input parameters or by increasing the maximum threshold of nodes in the training step. As the classification was performed on a per-event basis there was no speed-up found by increasing the number of trees in a decision forest.

It is important to note that events in the HEP dataset are independent and can be calculated in parallel. Hence limitations due to sample size can be easily overcome by scaling dataset processing horizontally across multiple GPU devices attached to the host (and by extension multiple GPU-enabled hosts) and by asynchronously staging data copying and kernel execution tasks with a pre-defined sample threshold equivalent to GPU device capacity.

![Classification Timing Performance (Tree Traversal)](image)

**Figure 3.** GPU and CPU timing performance comparison using the HEP dataset
5. Conclusions
The random forest classification method was investigated to assess whether GPUs were a viable option to improve the processing time of automatic detection of anatomical landmarks in three-dimensional medical imaging data. A feasibility study demonstrated that GPUs were capable of providing much faster classification than a CPU-only algorithm in the case where each decision tree could be treated as independent from the rest of the forest. Furthermore, the same GPU-based code was applied to a representative HEP dataset that demonstrated similar performance improvements independent of sample size up to a device-dependent threshold.

Offloading computationally intensive sections of machine learning classification techniques to many-core devices is considered to be more beneficial for higher volumes of input data and for increasing model complexity. No significant improvement would be observed if the classification is infrequently applied in a data analysis workflow or if the baseline CPU implementation was comparatively quick to complete. Here, the overhead from device initialisation and data movement (such as host to device transfer latency) would reduce the performance boost from parallelisation achieved by running on many-core devices.

Many-core implementations can offer a platform to improve discrimination capabilities of a given method by parallelising repeated applications of the training process to determine optimal input parameters. In addition, GPU kernels can be processed concurrently which would allow classification workloads to be multiplexed either on the same dataset or on disjointed workloads to provide a computationally efficient acceleration service for machine learning applications.

This study was a timely assessment into the benefits of many-core devices for large-scale data analysis given the increasing popularity of ensemble machine learning techniques used in HEP data analysis. A collaborative approach allowed understanding of the application of machine learning methods in a mature production environment in other scientific fields with ongoing studies into the adoption of many-core architectures to improve machine learning classification performance proving mutually beneficial.
References

[1] Antonio Criminisi and Jamie Shotton. *Decision forests for computer vision and medical image analysis*. Springer, 2013.

[2] Mohammad A. Dabbah, Sean Murphy, Hippolyte Pello, Romain Courbon, Erin Beveridge, Stewart Wiseman, Daniel Wyeth, and Ian Poole. Detection and location of 127 anatomical landmarks in diverse CT datasets. Proc. SPIE, 9034: 903415-903415-11, 2014. doi: 10.1117/12.2039157. URL http://dx.doi.org/10.1117/12.2039157.

[3] *Libxml2 project web page* [online] Available http://xmlsoft.org [accessed 15/05/2015]

[4] *CUDA parallel computing platform* [online] Available http://www.nvidia.com/object/cuda_home_new.html [accessed 15/05/2015]

[5] *Kitware Paraview* [online] Available http://www.paraview.org [accessed 15/05/2015]

[6] *Kaggle Higgs Boson Machine Learning Challenge* [online] Available https://www.kaggle.com/c/higgs-boson [accessed 15/05/2015]

[7] *Scikit Learn: Machine Learning in Python* [online] Available http://scikit-learn.org/stable/ [accessed 15/05/2015]