

Computational Intelligence on Graphics Processing Hardware

Ryan Meuth

Applied Computational Intelligence Laboratory,

Department of Electrical and Computer Engineering

University of Missouri-Rolla

1870 Miner Circle

Rolla MO, 65401

rmeuth@umr.edu

Keywords:

Graphics processing unit, neural networks, evolutionary algorithms, genetic algorithms, clustering, Adaptive Resonance Theory, SOM,

Modern graphics processing units (GPU) have a wider range of uses than the 3D graphics and computer games for which they are commonly known. From machine vision to finite element analysis, GPU's are being used in diverse applications, collectively called General Purpose Graphics Processor Utilization. In recent years consumer graphics processing units have experienced significant increases in performance, driven by more realistic game simulations and popular multimedia demands. As a result, the graphics industry has leveraged a parallel processing model which allows for a doubling of graphics computing capability every six months, as opposed to the 18 month doubling rate for general computing processors as illustrated in Figure 1. As graphics processors become more capable and flexible, they are increasingly desirable platforms for general computation.

Computational intelligence is a branch of artificial intelligence that uses with heuristic algorithms to solve optimization problems. Methods such as neural networks, clustering and evolutionary algorithms are all utilized in the computational intelligence field. For large problems, these methods can be highly computationally expensive thus it is appealing to port CI methods to the GPU. Algorithms from every category of computational intelligence methods have been implemented on graphics processors, from search methods to fuzzy systems, clustering methods and neural networks, even evolutionary algorithms.

Owens provides a comprehensive overview of the industry of general purpose computation on GPU's. However, Owens neglects much of the computational intelligence applications on graphics processing units. Here, an overview of these techniques is presented with associated challenges and limitations, as well as a new implementation of the Adaptive Resonance Theory clustering method on GPUs.

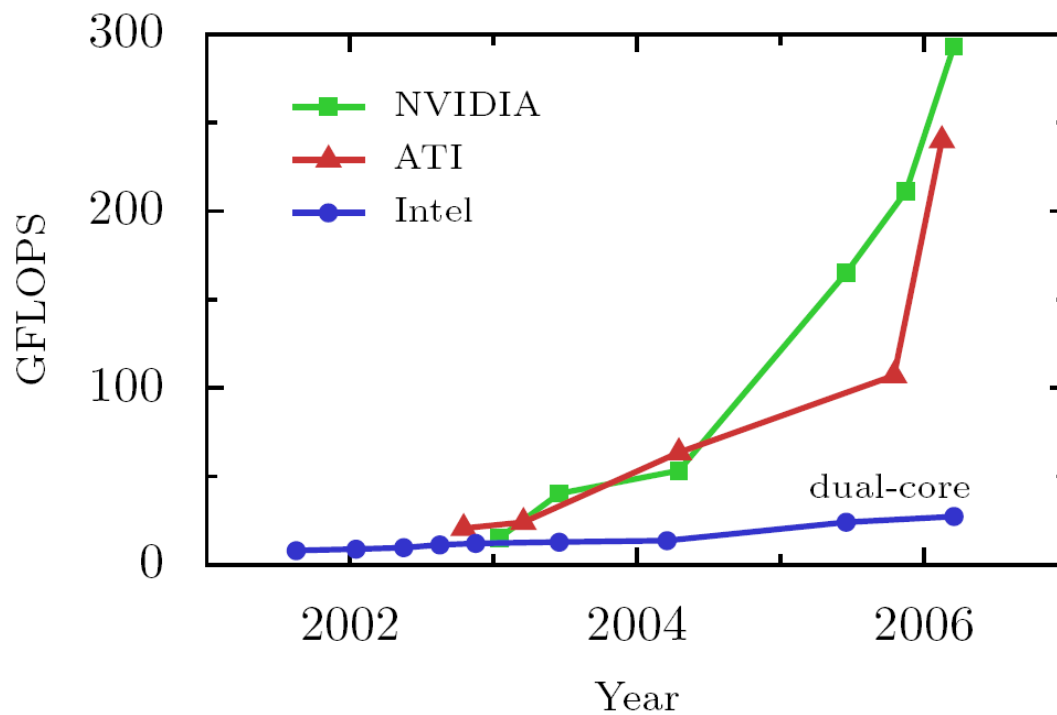


Figure 1. Shows the exponential increase in performance of graphics processing units compared to the performance of Intel processors over the last 4 years.

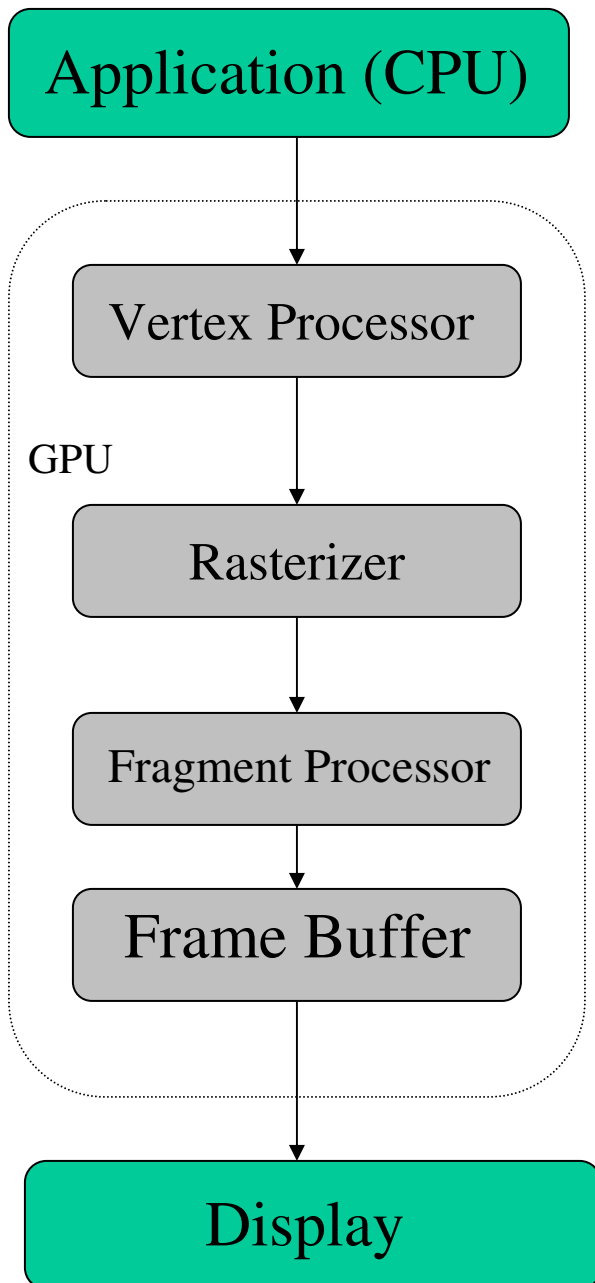


Figure 2 – The graphics processing pipeline. Modern GPU's combine the vertex and fragment processor into a unified shader unit that is able to perform either of these functions. Currently, GPU's can include up to 128 unified shader units.

Figure 2 shows an overview of the graphics processing pipeline. On the host system side, the application generates a data structure to be rendered, consisting of a set of vertexes

and their corresponding colors that define a polygon. This data structure is passed to the vertex processor, the first programmable unit in the graphics pipeline, which applies perspective transformations to the vertices. A rasterizer then maps these coordinates to pixel locations, generating a set of fragments. Fragments are then passed to the fragment processor, the second programmable unit in the pipeline. The fragment processor determines which fragments are to be drawn to the frame buffer, and then fills pixels with color information based on a program called a shader. Shader programs allow complex lighting and texture information to be mapped onto pixels. The frame buffer holds the completed image for output to a display.

To maintain high frame rates under increasingly graphically intensive applications the vertex and fragment processors have been implemented as a single-instruction, multiple-data parallel processing architecture. Modern graphics processors combine vertex and fragment processors into a generalized unified shader unit. At the time of this writing, GPU's can include up to 128 of these unified shader units, operating at up to 1.3Ghz. As the entire pipeline is based on the 32-bit floating point data type, this yields a significant processing capability on the order of hundreds of GFLOPS per processing unit.

Additionally, bus enhancements now allow multiple graphics cards to work together in the same system.

For general purpose computing, the GPU architecture lends itself well to applications in which the same calculations are repeatedly performed on large blocks of data. Particle systems, finite element analysis, image processing, and other numerical computations are

well suited to utilize the GPU. However, the shader units of GPU's do not yet include efficient branching hardware, making algorithms utilizing data-dependant operations difficult to implement efficiently. Additionally, the data bus that hosts the GPU is often inefficient for small data transfers, so to achieve a reasonable speedup data must be operated on in batches. Many of these difficulties have been overcome by creative algorithm design and implementation on the target systems. The widespread availability of these devices has allowed inexpensive and high performance computing environments to be constructed that leverage both CPU and GPU capability to create a 'cluster of clusters.'

GPU shader programs are written in a language similar to assembly, and can be developed through a graphics programming interface, such as OpenGL or DirectX. High-level languages such as Brook, Sh, and RapidMind allow developers to use C-based languages to write shader programs, providing data abstraction and useful functions, reducing the learning curve of these devices.

Search

Iterative search methods have been applied to GPU's, illustrating their usefulness in game AI methods, motion planning, and DNA sequence alignment. Lengyel details an algorithm for motion planning that is actually the slowest possible on a serial machine, but achieves optimal real time operation utilizing a GPU's parallel hardware. Voss uses

GPU hardware to accelerate the analysis of biological sequences, resulting in a speedup of up to 4.4 times over a modern Intel Pentium 4 processor.

Evolutionary Algorithms

Evolutionary Algorithms simulate the natural evolutionary process and apply it to the optimization of complex problems. An EA keeps a population of individuals, where each individual represents a possible solution to a given problem. Each individual is evaluated and assigned a fitness value, which determines the probability that the individual will be allowed to contribute its digital DNA to the next generation. Evolutionary algorithms, being a highly computing intensive methodology as well as inherently parallel and repetitious processes, can benefit greatly from graphics hardware acceleration. Wong shows a variety of methods for porting evolutionary programming and genetic algorithms to graphics hardware, achieving speedups of up to 5 times on problems with large population sizes.

Neural Networks

Artificial neural networks attempt to capture the adaptability of biological neurons in a mathematical model for information processing. ANNs are very powerful tools that are highly parallelizable but also computationally expensive and match well with the GPU computing architecture. As a workhorse of the computational intelligence field, there exists a high demand for this acceleration. Zhongwen achieves a massive 200 times increase in performance of a multilayer perceptron implemented on graphics hardware over a typical cpu, enabling real-time soccer ball tracking on commodity hardware.

Bernhard takes a different approach, implementing spiking neural networks for image segmentation, which achieves up to a 20 times increase in performance.

Unsupervised Learning

Unsupervised learning methods are concerned with grouping large bodies of data into self-similar categories. For example, separating mushrooms into species groups based on their features, without knowing the types of mushrooms beforehand. The repetitive operations of data clustering have allowed many clustering algorithms to be ported to the GPU, but the iterative and data-dependant nature of the algorithms have prevented large performance gains. Bohn implements Kohonen feature maps on GPU hardware. Even on graphics cards from '98, performance was improved by 5 times on large data sets using the GPU. Harris uses a combination of the GPU and CPU to implement Fuzzy C-means clustering, using the GPU to perform distance computations and the CPU to perform template updates. Using this method, Harris was able to triple the performance of the algorithm over the just the CPU alone.

Showing that even iterative methods can gain significant improvement when properly ported to graphics processing hardware, a new implementation of the Adaptive Resonance Theory data clustering method is detailed here. Created by Carpenter and Grossberg, Adaptive Resonance Theory or ART is an efficient method for clustering large streams of data into self-similar categories. In contrast to K-means which uses a fixed, pre-specified number of categories, the ART method adds categories automatically, depending on the properties of the data.

To implement the ART method on the GPU, 4 parallel functions were implemented, using the RapidMind development platform. These functions are distance, minimum with index, update template, and add a new template. The update template and add template methods are not typically expensive portions of the ART method, but they are implemented on the GPU side to minimize the number of data transfers necessary. The decision on whether to add a template or update a template is performed on the CPU, as this is a simple operation that does not require additional data transfers.

The distance function calculates the difference between an input data vector and the existing category templates, producing an array of match factors. This is the most computationally expensive portion of the ART algorithm, so the method benefits greatly from parallelization here.

The minimum with index function is used to find the category that best matches the input vector. This function uses a binary-tree based reduction method to find the minimum value in $\log(N)$ steps, where N is the number of elements to find a minimum in. This also significantly contributes to the acceleration of the algorithm, as the best that a sequential machine can perform a minimum function is in N steps.

The CPU side then decides whether to update the existing category or to add a new category based on a pre-specified threshold. The CPU compares the match factor of the closest category with this threshold. If the match factor is greater than the threshold, a new category is added, and the input data vector is used as the initial value for this template. If the match factor is less than the threshold, the category template is slightly modified to more closely match the input vector using the update template function.

Figure 3, which compares the performance of the ART method between implementations on an ATI Radeon X1600 GPU and an Intel CPU operating at 1.8GHz, illustrates that for large streams of vectors, the GPU implementation can have up to a 100 times speedup over the CPU implementation. This is due to both the parallel nature of the GPU hardware, and the fact that the math units of the GPU are highly optimized for floating point calculation.

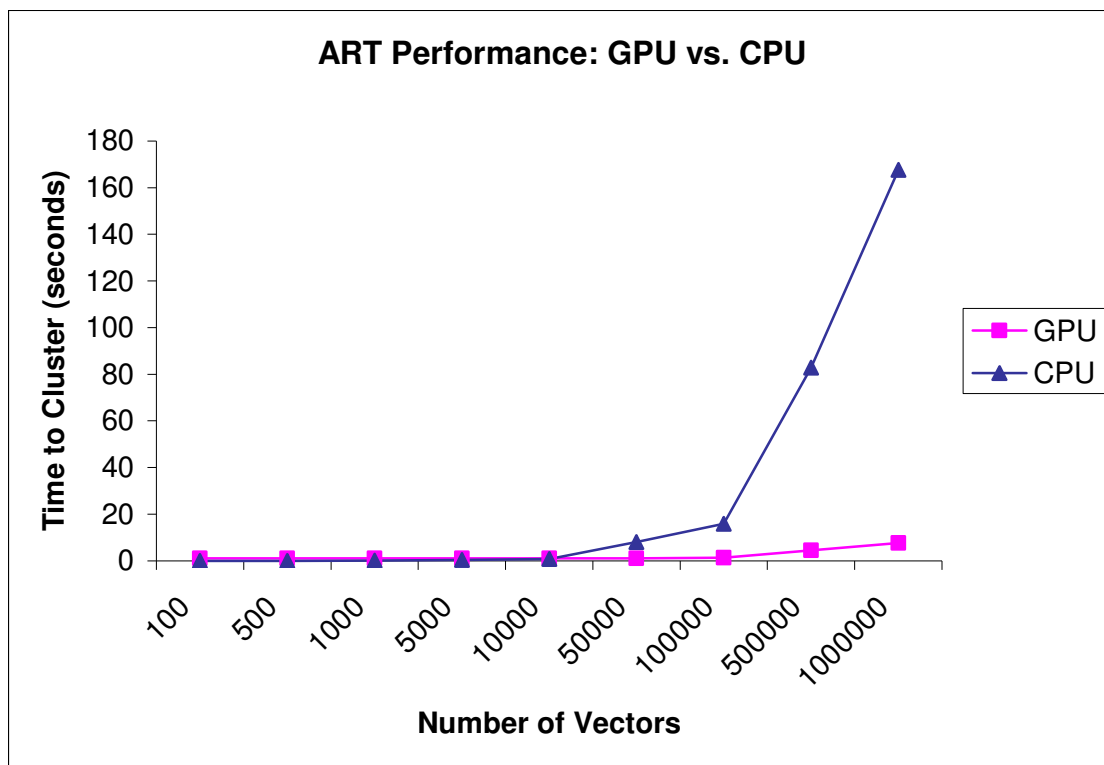


Figure 3. Comparing the Performance of ART implemented on GPU and CPU

Conclusion

As is shown by this review of the literature and research, significant performance gains can be elicited by implementing computational intelligence algorithms on graphics processing units. However, there is an amount of art to these implementations, as in some

cases the performance gains can be as high as 200 times, but as low as 2 times or actually less than CPU operation. Thus it is necessary to understand the limitations of the graphics processing hardware, and to take these limitations into account when developing algorithms targeted at the GPU. It should also be noted that all the reviewed papers in this document were operating on last-generation hardware. As of the end of 2006, the next generation graphics hardware has been released, which include an order of magnitude more shader units per processor, as well as improved branching capabilities. Consider the possible capability of 256 programmable shader units working in parallel at 1.3Ghz each, enabling the amount of computing power previously seen in large supercomputers, on a single desktop machine, at a fraction of the cost. Software designers now have extremely powerful computing tools at their disposal for entertainment, scientific computing, and computational intelligence applications.

References:

- [1] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, Timothy: A Survey of General-Purpose Computation on Graphics Hardware, J. Purcell Computer Graphics Forum, Volume 26
- [2] GeForce 8800 specifications, http://www.nvidia.com/page/geforce_8800.html, accessed November 9, 2006.

- [3] Lengyel J., Reichert M., Donald B. R., Greenberg D. P.: Real-time robot motion planning using rasterizing computer graphics hardware. In Computer Graphics (Proceedings of ACM SIGGRAPH 90) (Aug. 1990), vol. 24, pp. 327–335.
- [4] G. Voss, A. Schröder, W. Müller-Wittig, B. Schmidt, Using Graphics Hardware to Accelerate Biological Sequence Analysis, IEEE Tencon 2005, Melbourne, Australia, 2005
- [5] Harris, C.; Haines, K.; Iterative Solutions using Programmable Graphics Processing Units. Fuzzy Systems, 2005. FUZZ '05. The 14th IEEE International Conference on May 22-25, 2005 Page(s):12 – 18
- [6] BOHN C. A.: Kohonen feature mapping through graphics hardware. In Proceedings of the Joint Conference on Information Sciences (1998), vol. II, pp. 64–67.
- [7] K. L. Fok, T. T. Wong and M. L. Wong: " Evolutionary Computing on Consumer-Level Graphics Hardware", IEEE Intelligent Systems, to appear.
- [8] Zhongwen Luo; Hongzhi Liu; Xincui Wu: Artificial neural network computation on graphic process unit, IEEE International Joint Conference on Neural Networks, 2005. IJCNN '05. Proceedings. Volume 1, 31 July-4 Aug. 2005 Page(s):622 - 626
vol. 1

- [9] F. Bernhard and R. Keriven. Spiking neurons on GPUs. In International Conference on Computational Science. Workshop General purpose computation on graphics hardware (GPGPU): Methods, algorithms and applications, Reading, UK, May 2006.
- [10] G. Carpenter and S. Grossberg, “A massively parallel architecture for a self-organizing neural pattern recognition machine,” *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1987.