01 Aug 2007

Approximate Dynamic Programming and Neural Networks on Game Hardware

Ryan J. Meuth

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

APPROXIMATE DYNAMIC PROGRAMMING AND NEURAL NETWORKS ON GAME HARDWARE

Ryan J. Meuth, Donald C. Wunsch II
University of Missouri- Rolla
Dept. of Electrical & Computer Engineering
1870 Miner Circle,
Rolla, MO, 65401

***Abstract* - Modern graphics processing units (GPU) and game consoles are used for much more than simply 3D graphics applications and video games. From machine vision to finite element analysis, GPU's are being used in diverse applications, collectively called General Purpose computation onf Graphics Processor Units (GPGPU). Additionally, game consoles are entering the market of high performance computing as inexpensive nodes in computing clusters. This paper explores the capabilities and limitations of modern GPU's and game consoles, surveying the ADP and neural network technologies that can be applied to these devices.**

I. INTRODUCTION

In recent years consumer graphics processing hardware has experienced significant growth in performance driven by increasingly realistic game simulations and popular multimedia demands. As a result, the gaming industry has leveraged a parallel processing model to provide a doubling of graphics computing capability every six months, as opposed to the 18 month doubling rate of general computing processors, leading to a "Super-Moore's Law" trend that is illustrated in Figure 1. As these graphics processors become more capable and flexible, they have become desirable platforms for general computation. Owens [1] provides a extensive overview of the industry of general purpose computation on GPU's. However, Owens neglects to mention neural network and approximate dynamic programming applications on graphics processing units. Here, we provide an overview of these techniques, with associated challenges and limitations.

Figure 2 shows an overview of the graphics processing pipeline.

Manuscript received January 30, 2007. This work was supported in part by The Boeing Company, M. K. Finley Endowment, and the National Science Foundation.
Ryan Meuth is with the Department of Electrical and Computer Engineering at the University of Missouri – Rolla, Rolla MO, 65401, USA. (phone: 573-341-4521; e-mail: rmeuth@umr.edu).
Donald C. Wunsch II is with the Department of Electrical and Computer Engineering at the University of Missouri Rolla, Rolla, MO, 65401, USA. (e-mail: dwunsch@umr.edu).

On the host system side, the application generates a data structure to be rendered, consisting of a set of verticies and their corresponding colors that define a polygon. This data structure is passed to the vertex processor, which is the first programmable unit in the graphics pipeline, which typically applies transformations to the vertices. The rasterizer then maps these coordinates to pixel locations, generating a set of fragments. These fragments are then passed to the fragment processor, the second programmable unit in the pipeline.

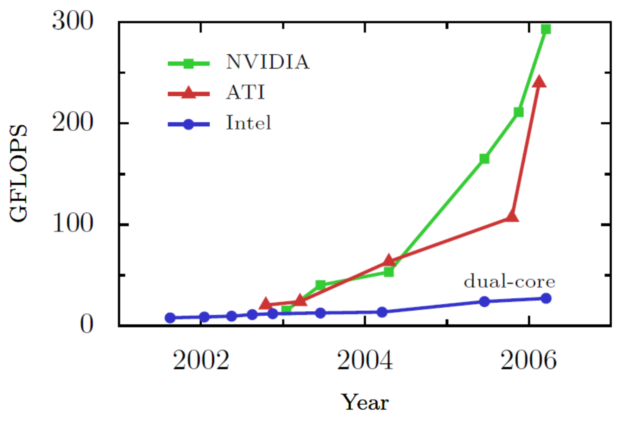

Fig. 1. The exponential increase in performance of graphics processing units compared to the performance of Intel processors over the last 4 years. Figure courtesy of Owens [1].

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 device.

To maintain high frame rates under increasingly graphically intensive applications the vertex and fragment processors have been implemented as a single-instruction, multiple-data (SIMD) parallel processing architecture. Modern graphics processors combine vertex and fragment processors into a generalized unified shader unit.

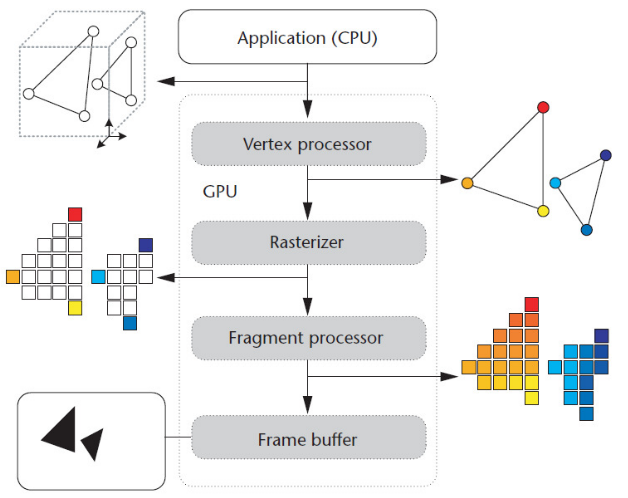

Fig. 2. The graphics processing pipeline. Modern GPU's combine the vertex and fragment processor into a unified shader unit that is able to perform either of these functions. Currently, GPU's can include up to 128 unified shader units. Figure courtesy of Goodnight [2].

At this writing, GPU's can include up to 128 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 in a single desktop chassis. Additionally, bus enhancements now allow multiple graphics cards to work together in the same system [3]. The widespread availability of these devices has allowed inexpensive high performance computing environments to be constructed that leverage both CPU and GPU capability to create a 'cluster of clusters' [4].

For general purpose computing, the GPU architecture lends itself well to applications where calculations are repeatedly performed on large blocks of data [5]. In this way, particle systems, finite element analysis, image processing, and other numerical computation are well suited to utilize the GPU. However, the shader units of GPU's do not yet include efficient looping or branching hardware, so algorithms utilizing data-dependant operations are difficult to implement effectively. Also, 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 [6].

Graphics processing units gain their computational power from their ability to apply a program to an array of vectors in parallel. Graphics processing units typically include processing pipelines that number in powers of two, thus the highest efficiency is achieved utilizing arrays that are similarly dimensioned. However, these calculations are limited to the data currently being operated on, meaning that there is no direct communication capability between the processing elements. Additionally, array sizes are fixed at compile-time, placing an upper bound on algorithms that require dynamic data sizes.

The lack of primitive looping and branching capability on GPUs decreases the efficiency of data dependant operations. Due to this limitation, each branch of an IF statement is evaluated, and only the result of the desired branch is retained, thus no computation can be saved using branching instructions. Additionally, the lack of explicit looping capability requires the un-rolling of algorithms into array or matrix form for efficient operation.

Many of these difficulties have been overcome by creative algorithm design and implementation on the target systems. Iterative, calculation intensive algorithms gain the most benefit from being ported to GPUs. Typically most of the porting difficulty involves mapping the algorithmic data structures into GPU video memory such that the data can be operated on efficiently, given the GPU limitations and capabilities.

GPU shader programs are written in a language similar to assembly, and can be developed through a graphics programming interface, such as the open-source OpenGL or Microsoft's DirectX. High-level languages such as Brook, Sh, and RapidMind allow developers to use C-based language extensions to create shader programs, providing data abstraction, reducing the learning curve of these devices. Some of these high level languages include library functions such as 'Reduce' which can perform a given operation on an array resulting in a single vector using an algorithm that operates with complexity of order $\log_2(N)$.

II. GAME CONSOLES

Driven by increasingly complex video games and graphics as well as new entertainment media demands such as internet, digital photography and video playback, consumer video game consoles have become powerful general purpose machines. At the same time, these systems must be brought to the public at an affordable price point. Figure 3 compares the ratio of Floating Point Operations per Second (FLOPS) per dollar of several game consoles and Intel Pentium Based Systems. Here we can see that the cost-effectiveness of the latest generation of game consoles is an order of magnitude higher than that of any Intel-based system. These features make gaming consoles a highly desirable platform for inexpensive high performance computing systems.

Though the performance per dollar ratio of these systems is attractive, they are not without limitations, most notably in their interconnect ability. Only the last two generations of consoles have included networking capabilities, and then only one port is provided, limiting the efficiency of interconnects architectures in console-based clusters.

Until the latest generation of game consoles, the technology embedded in consoles has often lagged behind the capability of personal computers at the time. However, the selling price of these devices makes them very competitive. In the previous generation of game consoles, this was recognized, and several attempts were made to utilize inexpensive game consoles as nodes in a super computing cluster. Very little success was made with the original Xbox [7], but researchers at the University of Illinois – Urbana Champaign succeeded in developing a 65-node computing cluster based off of the popular Playstation 2 video game console. This cluster was used for chemical simulations, and with a price point of $15,000 for the entire cluster, the system provided a high level of performance per dollar [8].

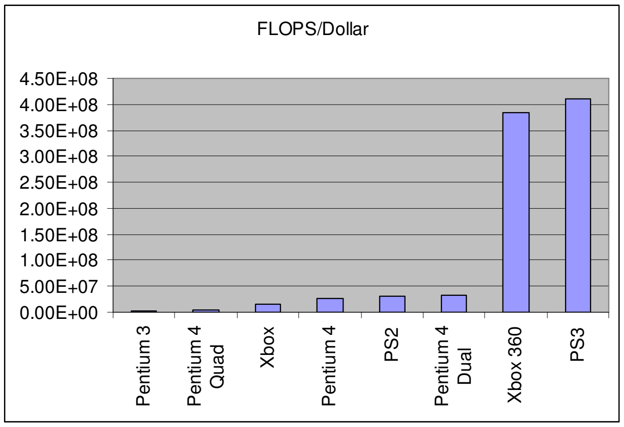

Fig. 3. Shows the FLOPS per dollar ratio of the past two generations of game consoles and Intel Processor-based systems. We can see that the latest generation of game consoles is several orders of magnitude more cost efficient than the latest Pentium-based systems.

The latest generation of game consoles differs from the former in that Microsoft's Xbox 360 and Sony's Playstation 3 both include new technologies that greatly surpass what is available in the home PC market. The Xbox 360 includes a tri-core Power PC processor operating at 3.2Ghz, theoretically providing a peak processing capability of 115.2 GFLOPS. The Xbox 360 surpasses PC-based computing capability by an order of magnitude, at a quarter of the cost [9]. Additionally, the Playstation 3 is capable of 205 GFLOPS provided by a nine-core processor called the Cell Broadband Engine cooperatively developed by Sony, IBM and Toshiba. The Cell consists of a single Power PC (PPE) based processor that manages 8 Synergistic Processing Elements (SPE) connected by an extremely high speed interconnect bus and shared memory. The PPE controls the SPEs like a cluster master node, implementing job queue, shared memory, and bus management. The Cell is unique in that the device is capable of managing 8 independent threads at full processor speed, with full branching and floating point operations available on each SPE [10, 11].

The Cell is also interesting in that it can be programmed using existing tools for graphics processing units, making much existing GPGPU work directly portable to these platforms.

Currently, no projects have been undertaken to develop high performance computing clusters based on the current generation game consoles. However, IBM will be using the Cell Processor in its next generation super computer, codenamed "RoadRunner." The machine will consist of 16,000 AMD Opteron cores matched with 16,000 Cell Broadband Engines, collectively rated at over 1 peta-FLOP/s. This will make it the most powerful super computer in the world by several orders of magnitude. It is to be built for the Department of Energy at Los Alamos National Laboratory in New Mexico. [12]

III. Dynamic Programming

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, and achieves optimal real time operation utilizing a GPU's parallel hardware on the uniform Piano Mover Problem [13]. Lengyel's algorithm first finds the action that should be taken at each location in the configuration space using a dynamic programming method to find the shortest path. This dynamic programming method has the characteristic of an expanding front of solutions, starting from the goal location, and proceeding through the search space. Once this action policy is found for every location in the space, the path is found from the starting position, and the system kinematics are modeled. This method operates optimally in real-time on reasonably sized problems utilizing the parallel computation properties of the GPU.

Most methods for finding the provably optimal policy of MDP's have an exponential time complexity as each state transition sequence must be evaluated. For large MDP formulations, it is often necessary to approximate the Bellman Equation. However, Todorov explores a sub-class of MDP's and details a method he calls Z-Learning for finding the optimal policy in linear time [14]. Similarly, Wunsch shows a closed-form solution to cellular simultaneous recurrent network adaptive critic design for the generalized maze problem [15]. For large problems, these algorithms can benefit significantly from the acceleration that GPU's can provide.

IV. Neural Networks

Neural networks are highly parallelizable and repetitious processes which should match well with the GPU computing architecture. As a workhorse of Approximate Dynamic Programming methods HDP, DHP, GDHP, etc., there exists a high demand for this acceleration. Zhongwen achieves a massive 200 times increase in performance of an MLP implementation on graphics hardware over a typical cpu, enabling real-time soccer ball tracking on commodity hardware [16]. Zhongwen uses the GPU to first extract a set of characteristics from image data, then applies a pre-trained MLP to these characteristics for classification. Zhongwen also provides several tips for ensuring efficient implementation of algorithms on GPU's. These tips include minimizing the pass count (or number of times a program must be applied to data), and minimizing data transfers between CPU and GPU sides.

Steinkraus implements a fully functional 2 layer artificial neural network on the GPU, resulting in a 3X speedup for both training and testing phases [17]. Steinkraus' method stores input data, training data, and weights in texture memory on the GPU. Then a batch training method is used to update the network weights. Steinkaus identifies 4 functions that he uses to compute the weight update, the least efficient of which is the Inner Product function. This function could be accelerated using the Reduce functionality of modern graphics processing units for additional efficiency.

Similarly, Chellapilla ports a convolutional neural network to the GPU, resulting in a 4x speedup [18].

Bernhard takes a different approach, implementing spiking neural networks for image segmentation, which achieves up to a 20 times increase in performance [19]. Bernhard utilized a special counter on the GPU called the Occlusion Query, which tracks how many times a pixel has been modified by a shader program. Using this counter, he was able to efficiently compile the activations of pre-synaptic neurons in a spiking neural network.

V. Conclusions

It can be easily seen from this review that significant performance gains can be elicited from implementing Approximate Dynamic Programming algorithms on graphics processing units. However, there is an amount of art to these implementations. 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. One can envision the possible capability of 256 programmable shader units working in parallel at 1.3 GHz each, in a single desktop box. Unfortunately none of the previous work has analyzed the performance of their algorithms relative to the number of computational units involved, which makes it unclear how new hardware will effect the performance of these methods.

Additionally, utilization of the latest generation game consoles can be a cost-effective method for accelerating ADP and neural network methods, and this application remains a topic of open exploration.

VI. 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," *Computer Graphics Forum*, Volume 26

[2] Goodnight, N., Wang, R., Humphreys, G., "Computation on programmable graphics hardware" *Computer Graphics and Applications*, IEEE Volume 25, Issue 5, Sept.-Oct. 2005 Page(s):12 – 15

[3] "GeForce 8800 specifications," http://www.nvidia.com/page/geforce_8800.html, accessed November 9, 2006.

[4] Zhe Fan; Feng Qiu; Kaufman, A.; Yoakum-Stover, S., "GPU Cluster for High Performance Computing" *Supercomputing*, 2004. Proceedings of the ACM/IEEE SC2004 Conference 2004 Page(s):47 – 47

[5] Ekman M., Warg F., Nilsson J., "An in-depth look at computer performance growth." *ACM SIGARCH Computer Architecture News* 33, 1 (Mar. 2005), 144–147.

[6] Tranoso, P.; Charalambous, M.; "Exploring graphics processor performance for general purpose applications," *Digital System Design*, 2005. Proceedings. 8th Euromicro Conference on 30 Aug.-3 Sept. 2005 Page(s):306 – 313

[7] "12 Node Xbox Linux Cluster," http://www.xl-cluster.org/index.php accessed January 19, 2007.

[8] "65 Node PS2 Linux Cluster," http://arrakis.ncsa.uiuc.edu/ps2/cluster.php accessed January 19, 2007. University of Illinois, Urbana-Champaign.

[9] "Comparison of Seventh Generation Game Consoles", www.wikipedia.com, http://en.wikipedia.org/wiki/Comparison_of_seventh-generation_game_consoles, Retrieved January 19, 2007.

[9] Pham D., Asano S., Bolliger M., Day M. N., Hofstee H. P., Johns C., Kahle J., Kameyama., Keaty J., Masubichi Y., Riley M.,

Shippy D., Stasiak D., Wang M., Warnock J., Weitzel S., Wendel D., Yamazaki T., Yazawa K.: “The design and implementation of a first-generation CELL processor.” *Proceedings of the International Solid-State Circuits Conference* (Feb. 2005), pp. 184–186.

[10] Brown, Jeffry (2005-12-06). “Application-customized CPU design.” IBM. http://www-128.ibm.com/developerworks/power/library/pa-fpfxbox/?ca=dgr-lnxw07XBoxDesign Retrieved on 2006-09-30.

[11] "Introduction to the Cell multiprocessor", *IBM Journal of Research and Development*, http://researchweb.watson.ibm.com/journal/rd/494/kahle.html, September 7, 2005.

[12] “IBM to Build World's First Cell Broadband Engine Based Supercomputer.” IBM http://www-03.ibm.com/press/us/en/pressrelease/20210.wss (2006-09-06). Retrieved on 2006-09-11.

[13] Lengyel, J., Reichert, M., Donald, B. R., Greenberg, D. P., “Real-time robot motion planning using rasterizing computer graphics hardware.” *Computer Graphics (Proceedings of ACM SIGGRAPH 90)* (Aug. 1990), vol. 24, pp. 327–335.

[14] Todorov, Emanuel, “Linearly Solvable Markov Decision Problems,” *Advances in Neural Information Processing Systems*, 2006. To Appear.

[15] Wunsch, D., “The cellular simultaneous recurrent network adaptive critic design for the generalized maze problem has a simple closed-form solution.” *IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000.* Volume 3, 24-27 July 2000 Page(s):79 - 82 vol.3

[15] Zhongwen Luo; Hongzhi Liu; Xincai 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

[17] Steinkraus, D.; Buck, I.; Simard, P.Y., “Using GPUs for machine learning algorithms,” *Proceedings of Eighth International Conference on Document Analysis and Recognition, 2005.* 29 Aug.-1 Sept. 2005 Page(s):1115 - 1120 Vol. 2

[18] Chellapilla K, Puri S, Simard P, “High Performance Convolutional Neural Networks for Document Processing,” *10th International Workshop on Frontiers in Handwriting Recognition* (IWFHR’2006) will be held in La Baule, France on October 23-26, 2006.

[19] F. Bernhard and R. Keriven. “Spiking neurons on GPUs,” *International Conference on Computational Science.* Workshop General purpose computation on graphics hardware (GPGPU): Methods, algorithms and applications, Reading, UK, May 2006.

Acknowledgement: The authors are grateful to Rui Xu for helpful discussions. This work was partially supported by the National Science Foundation and the M.K. Finley Missouri endowment.