SUPERCOMPUTING WITH TOYS: HARNESSING THE POWER OF NVIDIA 8800GTX AND PLAYSTATION 3 FOR BIOINFORMATICS PROBLEMS

Justin Wilson, Manhong Dai, Elvis Jakupovic, Stanley Watson and Fan Meng*

Molecular and Behavioral Neuroscience Institute and Department of Psychiatry, University of Michigan, Ann Arbor, MI 48109, United States of America

**Email: mengf@umich.edu*

Modern video cards and game consoles typically have much better performance to price ratios than that of general purpose CPUs. The parallel processing capabilities of game hardware are well-suited for high throughput biomedical data analysis. Our initial results suggest that game hardware is a cost-effective platform for some computationally demanding bioinformatics problems.

1. INTRODUCTION

Biomedical data analysis, visualization and mining demand more and more computing power in the post-genome era. Computer clusters are the prevailing solution for many bioinformatics laboratories and centers for accelerated large-scale data analysis. However, expanding the computing capacity of an existing cluster by more than an order of magnitude using traditional methods in a time of leveling-off processor speeds is difficult and expensive.

State-of-the-art game consoles and graphics processing units possess enormous computing power that can be directed at a variety of data analysis tasks[1–4]. However, the use of game hardware in bioinformatics is still rare and limited to special applications. The GPGPU website listed only one bioinformatics-related application, which reported a 2.7-fold speedup of the most time-consuming loop in the RAxML phylogenetic tree inference program when using a GeForce FX 5700 LE graphics card instead of a Pentium 4 3.2 GHz processor[5]. Most recently, the famous Folding@Home project developed clients for both ATI graphics processing units (GPU) and the Sony PlayStation 3 (PS3). In fact, PS3 already exceeds all participating computers in the number of TFLOPs contributed to the Folding@Home project[6].

A major obstacle to the wide-spread deployment of such promising game hardware was the lack of development tools. Traditionally, a developer had to learn a graphics API and cast their problem like a graphics problem in order to use a GPU for general computation. However, the recent release of the Compute Unified Device Architecture (CUDA) by NVIDIA has circumvented this problem and greatly facilitated developing software for NVIDIA GPUs[7]. In addition, the highly acclaimed Cell Broadband Engine (CBE) in the PS3, can be programmed using C instead of assembly with the free IBM Cell SDK[8]. Furthermore, third party vendors such as PeakStream[9] and RapidMind[10] allow the same program to be compiled and automatically optimized without modification for different multi-core platforms, thus greatly shortening the development cycle for different parallel computing platforms.

The computationally-intense nature of high-throughput data analysis led us to examine the possibility of utilizing game hardware to speed-up several common algorithms. Our results are very encouraging and we believe game hardware is an effective platform for many bioinformatics problems.

2. MATERIALS AND METHODS

Single or multiple CPU tests were performed on an 8x Opteron 865 (dual core) sever with 64G PC2700 memory running Fedora Core 2. GPU tests were performed on a 2x Opteron 275 (dual core) server with 4G of memory and a BFG GeForce 8800GTX with a core frequency of 600 MHz. The PS3 used in this project was a 60 GB version. The complier used for single and multiple Opteron core implementations was GCC 3.3.3. CUDA 0.8 and IBM Cell SDK

*Corresponding author.

2.0 were used for the 8800GTX and PS3 programs, respectively. We used RapidMind version 2.0 beta 3 and followed their write-once and run-anywhere paradigm for each platform. See our webpage[a] for the details of our tests.

3. RESULTS

3.1. 8800GTX and CBE vs. 1x and 16x CPU

Table 1 summarizes the performance of a single precision vector calculation when using the native development environments for the 8800GTX and CBE as well as the RapidMind platform. The calculation is described by

$$\vec{a}\prod^{N}\vec{b}\diamond\vec{b}$$

where $\diamond$ is an element-wise division operator, $\vec{a}$ and $\vec{b}$ are vectors of 9437184 elements and N is the number of repeated $\vec{b}\diamond\vec{b}$ calculations. The column headings for table 1 are as follows: "I" is the number of times the calculation was performed, "N" is the number of repeated $\vec{b}\diamond\vec{b}$ calculations, "1x" represents a single CPU, "16x" represents 16 CPUs, "GPU" represents the 8800GTX, "PS3" represents the CBE, and "RM" represents the designated hardware under the control of RapidMind.

Table 1. Vector multiplication/division performance on different platforms (seconds)

I	N	1x	16x	GPU	GPU RM	PS3	PS3 RM
10	500	426.6	32.3	2.2	2.5	96.4	
1	500	42.7	3.3	0.3	1.5	9.7	
10	100	79.1	6.2	1.3	1.9	19.6	574.6
1	100	7.9	0.6	0.2	0.7	2.0	559.5
100	10	56.7	7.1	11.7	13.2	21.4	11.5
10	10	5.7	0.7	1.2	1.9	2.2	6.1
1000	1	59.7	30.6	116.2	127.9	41.3	21.0
100	1	6.0	4.1	11.7	13.4	4.4	2.6
1000	0	19.7	15.0	106.0	127.8	20.0	17.0
100	0	2.0	1.6	10.2	13.2	2.3	2.2

Due to the physical design, game hardware does not provide an advantage for operations involving a large number of memory reads and writes (lower half of table 1). When a small number of memory operations (low iteration) are combined with CPU intensive operations (high calculation), the PS3 is more than 4 times the speed of a single Opteron 865 core. Most strikingly, a single NVIDIA 8800GTX is about 200 times faster than a single Opteron 865 core and more than 10 times faster than our 16-core (8x2) server. These results should be interpreted with the understanding that these numbers represent the upper limit of game hardware performance since the entire problem resided in the main memory of each device and there were no conditional statements.

Executables generated by RapidMind showed similar performance improvements on the 8800GTX when compared to executables generated using CUDA. The version we tried lacked optimization support for the CBE but RapidMind has promised such optimization in future versions[11]. Regardless, the ability to use the same source code for different multicore platforms should significantly help the adoption of game hardware.

3.2. Clustering Algorithms

Clustering is one of the most widely used approaches in bioinformatics. However, clustering algorithms are CPU intensive and a speedup would benefit problems ranging from gene expression analysis to document mining. A full clustering algorithm usually has two main components: determining the similarity of various samples (vectors) through a distance measure and the classification of samples into different groups through a clustering method[12]. We decided to implement two distance calculation methods, Euclidean and B-spline-based mutual information[13], and two clustering methods, single-link hierarchical clustering and the centroid k-means clustering for an 8800GTX, and investigate their performance under various conditions. The mentioned implementations have been used to generate similarity matrices and cluster documents from the MEDLINE database represented by MeSH term vectors and gene expression values from U133A GeneChips.

GPUs are best suited for parallel data processing with a high ratio of arithmetic to I/O and minimal amount of conditional instructions. Memory reads and writes between the host computer and GPU should be minimized. Data should be aligned

[a]http://wiki.mbni.med.umich.edu/wiki/index.php/Toycomputing

in memory and memory access patterns should be sequential and regular. A good strategy for design algorithms for the GPU is to examine the data dependency between the stages of an algorithm and have a kernel for each stage. Furthermore, having each thread or each block compute one independent element of the output of a stage automatically eliminates the need for synchronization between blocks.

Using these rules yields a distance matrix calculation kernel where each element in the distance matrix is computed by one block. First the vectors are copied to the device and aligned in memory. Each thread then computes the difference between two elements of two vectors and accumulates the results until both vectors are exhausted. Then, the shared memory between the threads can be utilized to sum up the contribution of each thread. Finally, the computed value is written to the distance matrix.

Finding the minimum in a distance matrix and updating values according to the Lance-Williams formula are both activities in hierarchical clustering that can be parallelized. Finding the minimum is similar to computing a distance matrix only the location of the minimum must be remembered. Updating the distance matrix can also be performed in parallel because only the rows and columns containing the two merged elements need to be updated. Consequently, one thread can process each column in the matrix.

The above techniques are also used in the k-means algorithm. The only seemingly difficult issue is adding up the vectors to calculate the new cluster centers. Since the GPU lacks atomic operations, having different blocks update the centers at the same time will not work correctly. However, by having each thread compute one element of one new cluster center, we circumvent the need for atomic operations. We also minimize number of memory reads by using the assignment matrix.

The computational speedup for calculating Euclidean distance matrices and mutual information matrices is presented in figure 1. The legend shows the number of elements in each vector and the type of calculation ("D" for distance, "B" for B-spline). The B-spline mutual information algorithm was configured to use 10 bins and spline order of 3^{13}. As expected, the B-spline mutual information matrix shows better GPU acceleration due to its higher arithmetic to I/O ratio. The figure also shows that it may not be worthwhile to perform small Euclidean distance calculations with a GPU since most of the processing time will be spent on memory operations.

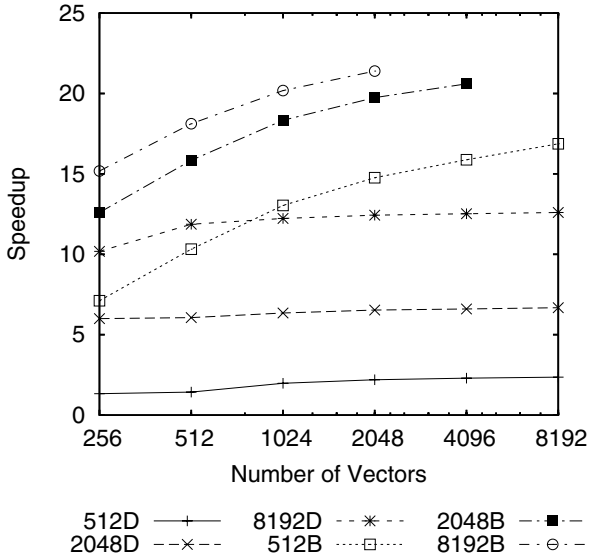

Fig. 1. Similarity Matrix Calculation Speedup

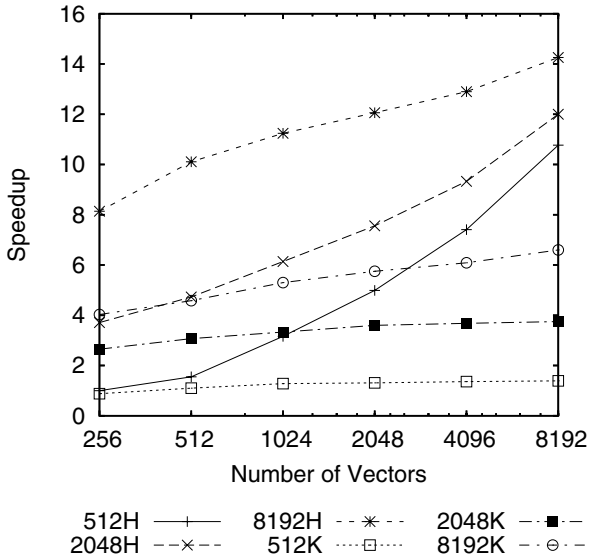

Fig. 2. Clustering Speedup

The computational speedup for hierarchical clustering ("H") (including the initial distance calculation) and k-means clustering ("K") is presented in figure 2. For k-means, the number of iterations was

fixed and the number of clusters was 4. As expected, both figures show that the speedup is strongly related to the dimensionality of the vectors to be classified because the elements of a data point can usually be operated on in parallel.

3.3. Monte Carlo Permutation

Permutation is widely used in statistical analysis but is often the most time consuming step in genome-wide data analysis. Table 2 compares the performance of an efficient Monte Carlo permutation procedure[14] for correlation calculation on different platforms using expression values from 4096 genes from 7226 U133A GeneChips deposited in the NCBI GEO database. It is obvious from the table that the 8800GTX can drastically speed-up Monte Carlo permutations without expanding an existing cluster given an open PCIe slot and adequate power supply.

Table 2. Monte Carlo permutation on game hardware (seconds)

Number	CPU (1x)	GPU	PS3
1	258.77	11.12	57.33
2	517.50	21.99	114.36
4	1035.01	43.75	228.60

4. DISCUSSION

Although we just started developing with game hardware, our results suggest that the NVIDIA GeForce 8800GTX is a very attractive co-processor capable of increasing the single precision floating point calculation speed by more than one order of magnitude in clustering and Monte Carlo permutation procedures. It is likely many other parallel data bioinformatics algorithms, particularly those related to high throughput genome-wide data analyses, will benefit from a port to game hardware.

Acknowledgments

The authors are members of the Pritzker Neuropsychiatric Disorders Research Consortium, which is supported by the Pritzker Neuropsychiatric Disorders Research Fund L.L.C. This work is also supported in part by the National Center for Integrated Biomedical Informatics through NIH grant 1U54DA021519-01A1 to the University of Michigan.

References

1. Angel E, Baxter B, Bolz J, Buck I, Carr N, Coombe et al. *http://www.gpgpu.org/* 2007.
2. Buck I, Foley T, Horn D, Sugerman J, Fatahalian K, Houston M, Hanrahan P. *ACM Transactions on Graphics* 2004; **23(3)**: 777-786.
3. Owens JD, Luebke D, Govindaraju N, Harris M, Krger J, Lefohn A E, Purcell TJ. *Computer Graphics Forum* 2007; **26(1)**: 80-113.
4. Mueller, F. *http://moss.csc.ncsu.edu/ mueller/cluster/ps3/* 2007.
5. Charalambous M, Trancoso P, Stamatakis A. *Proceedings of the 10th Panhellenic Conference on Informatics (PCI 2005), Springer LNCS* 2005; 415-425.
6. Folding@Home. *http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats* 2007.
7. NVIDIA. *http://developer.nvidia.com/object/cuda.html* 2007.
8. IBM. *http://www.alphaworks.ibm.com/tech/cellsw* 2006.
9. PeakStream. *http://www.peakstreaminc.com/product/overview/* 2006.
10. RapidMind. *http://www.rapidmind.net/* 2006.
11. RapidMind. *http://www.rapidmind.net/ pdfs/RapidMindCellPorting.pdf* 2006.
12. Wikipedia. *http://en.wikipedia.org/wiki/Data_clustering* 2007.
13. Daub CO, Steuer R, Selbig J, Kloska S. *BMC Bioinformatics* 2004; **5**: 118.
14. Lin, DY. *Bioinformatics* 2005; **21(6)**: 781-787.