

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 tasks¹⁻⁴. 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 processor⁵. 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 project⁶.

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 GPUs⁷. 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 SDK⁸. Furthermore, third party vendors such as PeakStream⁹ and RapidMind¹⁰ 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 compiler used for single and multiple Opteron core implementations was GCC 3.3.3. CUDA 0.8 and IBM Cell SDK

*Corresponding author.

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.

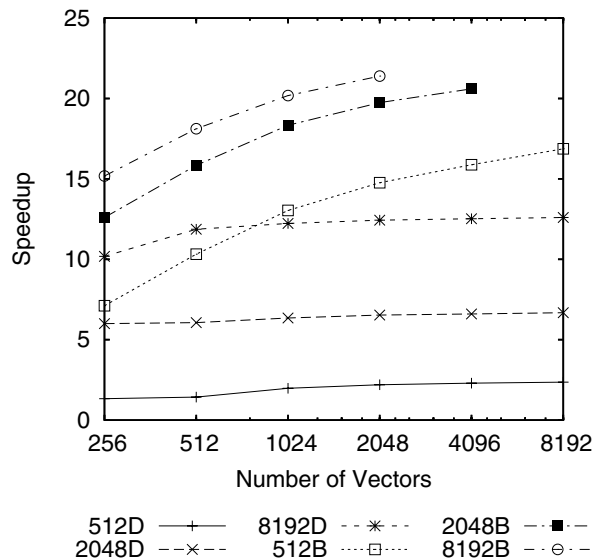


Fig. 1. Similarity Matrix Calculation Speedup

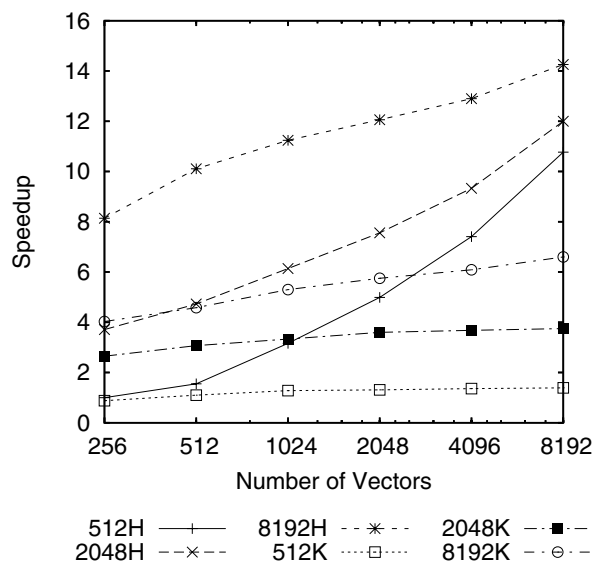


Fig. 2. Clustering Speedup

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

