Applying GPU Dynamic Parallelism to High-Performance Normalization of Gene Expressions

Roberto P. Souto[1], Carla Osthoff[1], Ana T. R. de Vasconcelos[1], Douglas A. Augusto[1], Pedro L. da Silva Dias, Andres Rodriguez[2], Oswaldo Trelles[2], Manuel Ujaldon[2]
[1]National Laboratory for Scientific Computing, [2]University of Malaga
email: {rpsouto,osthoff,atrv,douglas,pldsdias}@lncc.br, {andres,ots,ujaldon}@uma.es

1 Introduction

High-density oligonucleotide microarrays allow several millions of genetic markers in a single experiment to be observed. Current bioinformatics tools for gene expression quantile data normalization are unable to process such huge data sets. In parallel with this reality, the huge volume of molecular data produced by current high-throughput technologies in modern molecular biology has increased at a similar pace, challenging our capacity to process and understand data. The arrival of CUDA has unveiled the extraordinary power of Graphics Processing Units (GPUs) to accelerate data intensive general purpose computing. In this work we have evaluated the use of dynamic parallelism for ordering gene expression data, where the management of kernels launching can be done not only by the host, but also by the device. We have compared the performance of the sequential quicksort algorithm from the GNU C Library (`glibc`) with the parallel implementation available in the CUDA-5.5 Toolkit Samples.

2 Q-norm: an algorithm for normalizing oligonucleotides

The ultimate goal of the Q-norm method is to make the distribution of probe intensities for each array in a set of samples the same [2], under the assumption that there is an underlying common distribution for all those samples. This suggests that one could give two disparate data sets the same distribution by transforming the quantiles of each data set to have the same value, which will be their average value. Extending this idea to N dimensions gives us a method of finding a common distribution from multiple data vectors.

Let $q_k = (q_{k_1}, ..., q_{k_1})$ for $k = 1, ..., nG$ be the vector of the k_{th} quantiles for all nE array experiments/samples of length nG which compose matrix X of dimensions $nG \times nE$ where each sample is a column. The quantile normalization goes as follows:

1. Sort each column of X to give X_{sort};
2. Take the means across rows of X_{sort} and assign this mean to each element in the row to get $\mathbf{x}_{avg}$ sort vector;
3. Produce X_{norm} as output by rearranging $\mathbf{x}_{avg}$ sort vector to have the same ordering as original X.

This method forces the values of quantiles to be equal, which may cause problems in the tails where it is possible for a probe to have the same value across all the array samples. However, this case is unrealistic since probeset expression measures are typically computed using the value of multiple probes.

3 Recursive QuickSort Algorithm and Dynamic Parallelism

The quicksort algorithm is one of the most popular in the task of sorting the data. It is an algorithm that employs a strategy of divide-and-conquer, that consists of recursively partitioning the data list according to a choice criterion of a pivot element. Having selected a list element pivot, it is divided into two parts: the left part with the elements smaller than or equal to the pivot, and the right part with the elements larger than the pivot. And this is done successively and may be done recursively. However, in previous GPU architectures to Kepler, each new call to quicksort kernel should be performed from the host (CPU). The dynamic parallelism introduced by K20 GPU allows new calls to quicksort are made from the device itself, no longer needed to be made by the host. With this, it is maintained recursion inherent quicksort algorithm, boosting performance.

4 Results

This section presents the computational performance of the Q-norm method using the quicksort algorithm from the GNU C Library (`glibc`)-`qsort` function as defined in `stdlib.h`-and also the GPU implementation from the NVIDIA CUDA-5.5 Toolkit Samples [5].

In both CPU and GPU implementations, the quicksort algorithm receives as input to be sorted an array of `struct` containing information of the value and original index of each element of the data set. It was done this way so that it is possible to recover the original position of the data after obtaining the average of all sorted experiments.

4.1 The input data set

The input data set was taken from the GEO (Gene Expression Omnibus) Web repository as submitted by Affymetrix under the GeneChip Human Mapping 500K Array Set (platform GPL3718).

The high computational cost and memory requirements of the Q-norm method result from its application to huge data sources, which in our case means arrays composed of more than 6 million gene expression values.

Each array contains positive integers values that were obtained by high-density oligonucleotide microarray technology provided by the Affymetrix GeneChip infrastructure [6], which is widely used in many areas of biomedical research.

Those integers are the target numbers to normalize, usually by means of some kind of averaging over every array element placed in the same quantile [3].

In the experimental case of this work, it is considered $nE = 32$ samples (number of arrays), with each one composed of $nG = 6,553,600$ gene expression values.

4.2 Performance Analysis

The experiments were carried out by first using exclusively the CPU in serial mode (a single core) for all stages of Q-norm, and later a mixed approach in which only the demanding sorting stage computed by quicksort is executed in parallel on the GPU (the other stages remain on the CPU, in sequential).

The configuration of computing resources was a CPU Intel Xeon E5650 and and a GPU NVIDIA Tesla K20c.

Table 1: Sort algorithms used

Agorithm	Function	Library
qsort^{cpu}	`qsort`	GNU glibc library
qsort^{gpu}	`quicksort_cdp`	CUDA-5.5 samples

According to Table 2, the speed-up achieved by the quicksort routine alone was about **22.8 times faster** than the exclusively serial code.

However, when we also consider the cost of data transfer between CPU and GPU (cudaMemcpy function), this gain drops to about **14.1 times faster**.

Moreover, when considering the total execution time of the method Q-norm, the speed-up is further reduced to about **4.8 times**.

Table 2: Q-norm performance of CPU and GPU sorting algorithms

	Time in seconds	
	CPU	GPU
	qsort^{cpu}	qsort^{gpu}
QNormMain	33.69	6.80
LoadFile	0.26	0.38
sortingAlgorithm	**29.51**	**1.30**
AccumulateRow	0.31	0.25
backPos	2.06	2.45
StoreFile	0.49	0.45
cudaMemcpy	-	0.80

5 Final Remarks

This work has presented methods for computing a quantile normalization of high-density oligonucleotide array data on GPUs. Our approach focuses on CUDA-5.5, which allows for exploiting dynamic parallelism offered by the GPU Kepler architecture. The results show that in the studied application the GPU parallel version with dynamic parallelism attains good speed-ups in the data sorting step. For future work we plan to evaluate the following hybrid parallelization strategies to achieve an effective overall speed-up considering the performance of the whole application, using strategies combining MPI+GPU or OpenMP+GPU. Moreover, in order to better evaluate the impact of dynamic parallelism, we can also adapt the original parallel recursive quicksort of the CUDA Toolkit Samples, to a non-recursive version of it. Thus, new quicksort launches are done only from the host.

Acknowledgments

The authors thanks to Brazilian Research Agency - CNPq (301877/2013-0), FAPERJ (E-26/102.025/2009) and CUDA Teaching Center Program.

References

[1] The GeneChip Human Mapping 500K Array data set Submitted to GEO by Affymetrix. http://www.ncbi.nlm.nih.gov/geo/

[2] B.M. Bolstad, R.A. Irizarry, M. Astrand, and T.P. Speed. *A Comparison of Normalization Methods for High Density Oligonucleotide Array Data based on Variance and Bias.* Bioinformatics, 19(2):185-193, 2003.

[3] R.A. Irizarry, B. Hobbs, F. Colin, Y.D. Beazer-Barclay, K. Antonellis, U. Scherf, and T.P. *Speed. Exploration, Normalization and Summaries of High Density Oligonucleotide Array Probe Level Data.* Biostatistics, 4(2):249?264, 2003.

[4] J.M. Mateos-Duran, P. Prins, A. Rodriguez, and O. Trelles. *Q-norm: A library of parallel methods for gene-expression q-normalization.* In Bioinformatics Open Source Conference, Stockholm (Sweden), June 2009.

[5] NVIDIA. *CUDA Samples - CUDA Toolkit Documentation.* http://docs.nvidia.com/cuda/cuda-samples/index.html#advanced-quicksort–cuda-dynamic-parallelism

[6] J.A. Warrington, S. Dee, and M. Trulson. *Large-Scale Genomic Analysis Using Affymetrix GeneChip,* chapter 6, pages 119?148. Microarray Biochip Technologies. BioTechniques Books, New York, USA, 2000