BIOINFORMATICS APPLICATIONS NOTE

Vol. 27 no. 2 2011, pages 284–285
doi:10.1093/bioinformatics/btq638

Genetics and population analysis

Advance Access publication November 25, 2010

Real-world comparison of CPU and GPU implementations of SNPrank: a network analysis tool for GWAS

Nicholas A. Davis, Ahwan Pandey and B. A. McKinney*

Department of Mathematical and Computer Sciences, University of Tulsa, Tulsa, OK 74104, USA

Associate Editor: Jeffrey Barrett

ABSTRACT

Motivation: Bioinformatics researchers have a variety of programming languages and architectures at their disposal, and recent advances in graphics processing unit (GPU) computing have added a promising new option. However, many performance comparisons inflate the actual advantages of GPU technology. In this study, we carry out a realistic performance evaluation of SNPrank, a network centrality algorithm that ranks single nucleotide polymorhisms (SNPs) based on their importance in the context of a phenotype-specific interaction network. Our goal is to identify the best computational engine for the SNPrank web application and to provide a variety of well-tested implementations of SNPrank for Bioinformaticists to integrate into their research.

Results: Using SNP data from the Wellcome Trust Case Control Consortium genome-wide association study of Bipolar Disorder, we compare multiple SNPrank implementations, including Python, Matlab and Java as well as CPU versus GPU implementations. When compared with naïve, single-threaded CPU implementations, the GPU yields a large improvement in the execution time. However, with comparable effort, multi-threaded CPU implementations negate the apparent advantage of GPU implementations.

Availability: The SNPrank code is open source and available at http://insilico.utulsa.edu/snprank.

Contact: brett.mckinney@gmail.com

Received on September 14, 2010; revised on November 3, 2010; accepted on November 8, 2010

1 INTRODUCTION

Previously, we developed a new eigenvector centrality algorithm called SNPrank (Davis *et al.*, 2010), which ranks interacting single nucleotide polymorphisms (SNPs) in a genetic association interaction network (GAIN) (McKinney *et al.*, 2009). Each SNP is ranked according to its overall contribution to the phenotype, including its main effect and second- and higher-order gene–gene interactions. GAIN and SNPrank provide a data-driven, network-based approach to identify important hub SNPs through conditional dependence with other SNPs and the phenotype, which allows for phenotype-specific pathway discovery that is not possible with more myopic approaches.

In the development process of SNPrank, we attempted to systematically address some of the typical implementation issues facing bioinformaticists plus emerging issues introduced by GPU. GPUs have demonstrated performance benefits in some bioinformatics applications (Sinnott-Armstrong *et al.*, 2009); however, many GPU comparisons do not reflect conditions in real-world applications. Thus, with the availability of GPUs and the numerical improvements made to Matlab and Python by third-party developers, we find it important to address implementation comparisons of SNPrank in a manner that accounts for all aspects of the computation. Moreover, with the emergence of deep sequencing technologies, optimization of many bioinformatics algorithms will need to be revisited.

2 METHODS

The SNPrank algorithm was written in each of Matlab, Python and Java using the same design patterns apart from the language syntax disparities. While Matlab provides comprehensive matrix functionality, general languages typically lack an integrated linear algebra component for efficient matrix computations. Thus, external language-specific libraries were employed: NumPy for Python and jblas for Java. All implementations use Basic Linear Algebra Subprograms (BLAS) as the underlying low-level linear algebra library. Additionally, Python and Java take advantage of an optimized BLAS library implementation called Automatically Tuned Linear Algebra Software (ATLAS).

Profiling the SNPrank algorithm revealed matrix computation as the largest bottleneck. As matrix multiplication is highly efficient on the GPU, this was the logical candidate for optimization. This augmentation required the use of CUDA, a widely used parallel computing architecture developed by NVIDIA. We opted for a streamlined approach for more rapid development, rather than write a custom kernel for CUDA matrix multiplication. We used a Python matrix library called CUDAMat, which allows for standard matrix calculations on the GPU in the same vein as NumPy. We used a similar, but commercial, add-on called Jacket for GPU computations within Matlab. Since Java lacks a suitable matrix GPU library, we only used Python and Matlab to test the GPU. Within the SNPrank algorithm, the Transition matrix T, described by Equation (5) in Davis *et al.* (2010), is calculated immediately in the CPU implementation, whereas the GPU must also communicate with the CPU host. The heavy lifting (i.e. matrix multiplication) is calculated on the GPU and copied over to the host where the rest of the equation is computed to produce a result.

The Wellcome Trust Case Control Consortium (WTCCC) Bipolar Disorder (BD) genome-wide association study (GWAS) was used to compare performance (Burton *et al.*, 2007). The initial list of 500k SNPs was filtered to about 215k using the PLINK linkage disequilibrium (LD) pruning option (Purcell *et al.*, 2007). The purpose of LD pruning here is to obtain a maximally independent set of SNP markers for SNPrank analysis. However, we note that the influence of LD on network centrality algorithms is an important area that is largely unexplored. PLINK was subsequently used to sort the SNPs by their *P*-values, and then the best 100, and best 1000 through 10 000 in increments of 1000 were saved to separate files. The GAIN algorithm was then applied to each file to produce a matrix encoding of the SNP–SNP interaction network (GAIN file) (McKinney *et al.*, 2009).

*To whom correspondence should be addressed.

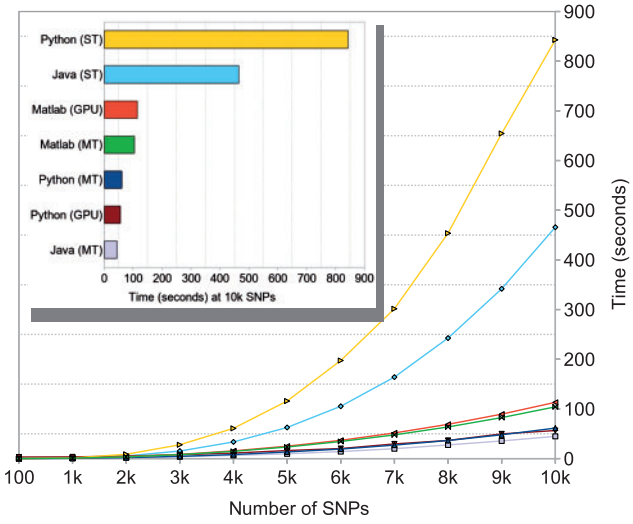

Fig. 1. Total execution times of SNPrank implementations for WTCCC BD data versus number of SNPs included in the analysis. Single-threaded (ST) and multi-threaded (MT) refer to CPU implementations unless GPU is specifically designated.

A GAIN file is a tab- or comma-separated values (CSV) file representing an $n \times n$ symmetric matrix, where n is the number of SNPs. Finally, SNPrank was applied to each GAIN network from each filtering scenario and implementations were timed from file input to final returned output in order to reproduce a typical analytical pipeline.

Our testbed consists of a GPU server with 48 GB of system RAM and two Tesla M1060 GPUs, each with 240 processing cores and 4 GB of RAM. The CPU is a six-core Intel Xeon X5650 running at 2.67 GHz. Only one GPU was utilized for this benchmark. The OS is 64-bit Ubuntu Linux 10.04.

3 RESULTS

Figure 1 illustrates the clear performance gain by the Python GPU over the corresponding single-threaded CPU implementation, with the gain increasing with the number of SNPs. However, the multi-threaded optimized CPU Python performance is practically the same as the GPU. For the top 10k SNPs, the multi-threaded CPU shows a 14X improvement over single-threaded CPU, and the GPU version is only 1.1X faster than the multi-threaded CPU version. Though it is difficult to discern from the scale of Figure 1, we note that the CPU is almost always faster than GPU for smaller datasets because the GPU requires a larger dataset to overcome the latency issues discussed below.

As Matlab is a multi-threaded environment by default, it did not require additional third-party packages. It was still slower than our fastest Python implementation, spending most of the execution time reading in the GAIN file. The Matlab GPU implementation proved slightly slower, largely due to the overhead involved in transferring data to and from the GPU. Java on the CPU uses the jblas library optimized for multi-threading and proved to be the fastest among the three languages, even besting the Python GPU implementation by almost 10 s for the top 10k SNPs, as shown in Figure 1.

4 DISCUSSION

By default, the third-party numerical libraries NumPy and jblas employ a single-threaded approach, which allows a naïve CPU implementation to be easily outperformed by GPU. However, recompiling the CPU libraries with multi-threading support narrowed the advantage significantly. The performance comparisons are not based upon raw matrix calculations, but rather on a real-world application of our algorithm from beginning to end.

Matlab provides an integrated environment for numerical calculations and includes an efficient computation engine. In contrast, Python and Java require some additional effort to take advantage of external numeric libraries, particularly for multi-threaded implementations. Unlike Matlab, both Java and Python are freely available and open source. Thus, cost and skill level are determining factors for the appropriate language used to implement algorithms and develop bioinformatics tools. A related benefit of their larger developer communities is that both Python and Java have more third-party tools and libraries, compared with Matlab. Moreover, while Jacket facilitates the conversion of Matlab code to GPU, this implementation actually showed slightly slower overall execution time.

Presently, GPU computations involve a large overhead of copying data to and from the device, which cannot be neglected in real-world applications. These bandwidth constraints of the PCIe interface contribute to a significant proportion of the total execution time (Datta *et al.*, 2008). Thus, resolving the latency issues in future designs will be necessary to increase the utility of GPUs for bioinformatics applications.

We limited our network analysis to the top 10 000 SNPs because it is unlikely that SNPs beyond this will contribute significantly to the genetic risk. An advantage of our comparison of SNPrank implementations is that performance was tested in the context of a real dataset. In a future publication, we will discuss the biological implications of our network analysis of the BD GWAS.

ACKNOWLEDGEMENTS

We would like to thank Chris Johnson for his programming contributions, and we would like to thank Nicholas Sinnott-Armstrong and Jason Moore for their helpful discussions.

Funding: National Institutes of Health [Grants no. K25 AI-64625 (PI: McKinney); R56 AI-80932 (PI: McKinney)].

Conflict of Interest: none declared.

REFERENCES

Burton,P.R. *et al.* (2007) Genome-wide association study of 14 000 cases of seven common diseases and 3000 shared controls. *Nature*, **447**, 661–78.

Datta,K. *et al.* (2008) Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, TX, pp. 1–12.

Davis,N.A. *et al.* (2010) Surfing a genetic association interaction network to identify modulators of antibody response to smallpox vaccine. *Genes Immun.* [Epub ahead of print; doi:10.1038/gene.2010.37].

McKinney,B.A. *et al.* (2009) Capturing the spectrum of interaction effects in genetic association studies by simulated evaporative cooling network analysis. *PLoS Genet.*, **5**, e1000432.

Purcell,S. *et al.* (2007) PLINK: A tool set for whole-genome association and population-based linkage analyses. *Am. J. Hum. Genet.*, **81**, 559–575.

Sinnott-Armstrong,N.A. *et al.* (2009) Accelerating epistasis analysis in human genetics with consumer graphics hardware. *BMC Res. Notes*, **2**, 149.