Enabling GPU Computing in the R Statistical Environment

Josh Buckner[1], Manhong Dai[1], Brian Athey[1,2], Stanley Watson[1] and Fan Meng[1,2]
[1]Molecular & Behavioral Neuroscience Institute and Psychiatry Department
[2]National Center for Integrative Biomedical Informatics
University of Michigan, Ann Arbor, MI 48109, US

Abstract

R is the most popular open source statistical environment in the biomedical research community. However, most of the popular R function implementations involve no parallelism and they can only be executed as separate instances on multicore or cluster hardware for large data-parallel analysis tasks. The arrival of modern graphic processing units (GPUs) with user friendly programming tools, such as nVidia's CUDA toolkit (http://www.nvidia.com/cuda), provides a possibility of increasing the computational efficiency of many common tasks by more than one order of magnitude (http://gpgpu.org/). However, most R users are not trained to program a GPU, a key obstacle for the widespread adoption of GPUs in biomedical research. To overcome this obstacle, we decided to devote efforts for moving frequently used R functions in our work to the GPU using CUDA. In the ideal solution, if a CUDA compatible GPU and driver is present on a user's machine, the user may only need to prefix "gpu" to the original function name to take advantage of the GPU implementation of the corresponding R function. We take achieving this ideal as one of our primary goals so that any biomedical researcher can harness the computational power of a GPU using a familiar tool. Since our code is open source, researchers may customize the R interfaces to their particular needs. In addition, because CUDA uses shared libraries and unobtrusive extensions to the C programming language, any experienced C programmer can easily customize the underlying code.

Implementation

Using the CUDA extension to C and the shared linear algebra library CUBLAS, we have implemented a variety of statistical analysis functions with R interfaces that execute with different degrees of parallelism on a Graphics Processing Unit (GPU). If an algorithm is comprised of common vector or matrix operations each performed once, we involve the GPU by implementing those operations with calls to CUBLAS. If an algorithm involves computing the elements of a large matrix, we can often merely assign each thread executing on the GPU a portion of a row and/or column. Algorithms for which we have implemented GPU enabled versions include the calculations of distances between sets of points (R dist function), hierarchical clustering (R hclust function). Pearson and Kendall correlation coefficients (similar to R cor function), and the Granger test (granger.test in the R MSBVAR package).

Results

Fig. 1: Speedup (R GPU vs. CPU)

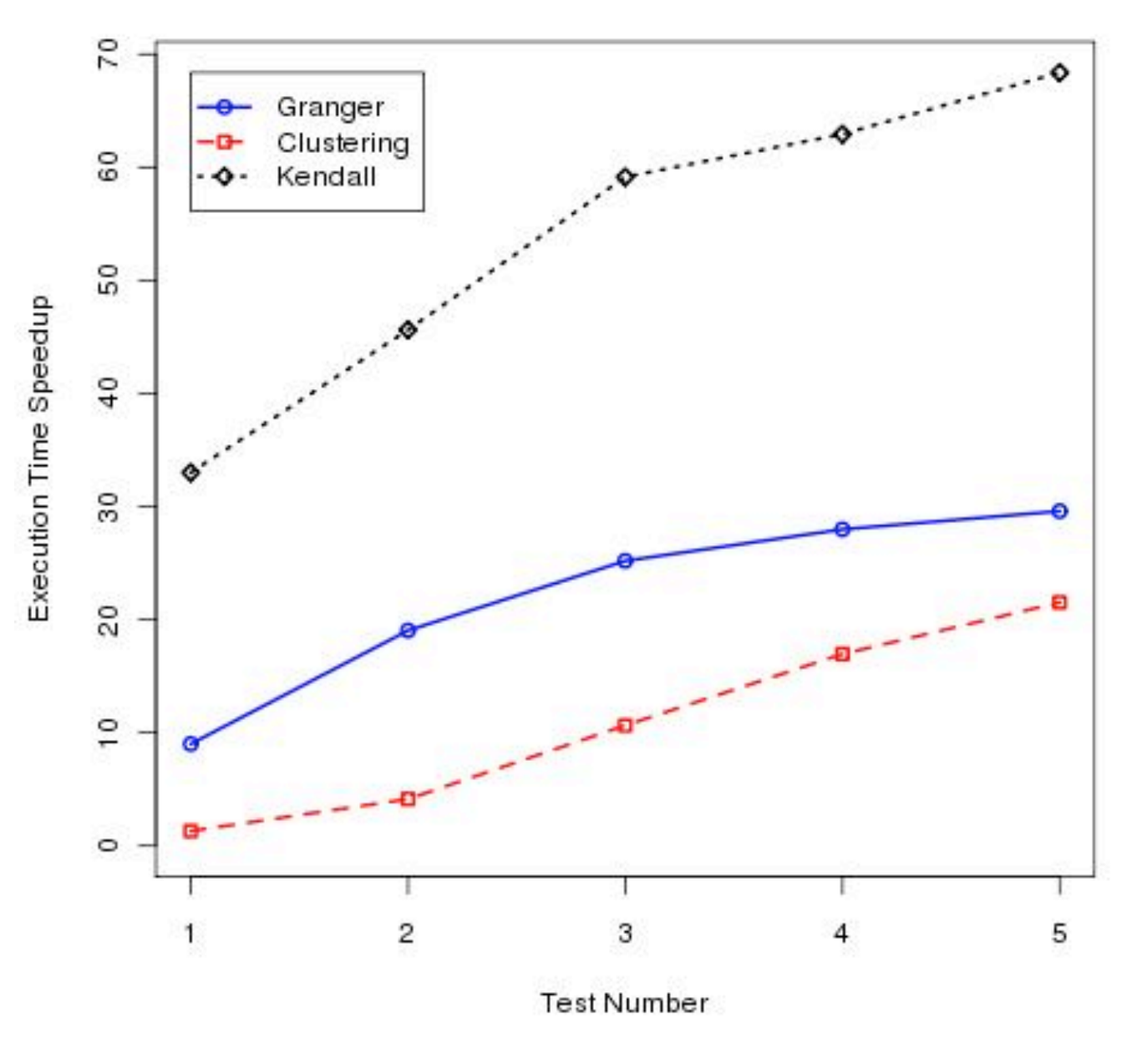

Figure 1 illustrates performance comparisons between four thread data parallel solutions using traditional R functions a on Intel Core i7 920 and our GPU enabled R functions using a GTX 295 GPU. The trials consisted of testing each of three algorithms with five randomly generated data sets. The Granger causality algorithm was tested with a lag of 2 for 200, 400, 600, 800, and 1000 random variables with 10 observations each. Euclidean distance and complete hierarchical clustering was tested with 1000, 2000, 4000, 6000, and 8000 points. Kendall's correlation coefficient was tested with 20, 30, 40, 50, and 60 random variables with 10000 observations each.

Fig. 2: Granger Times

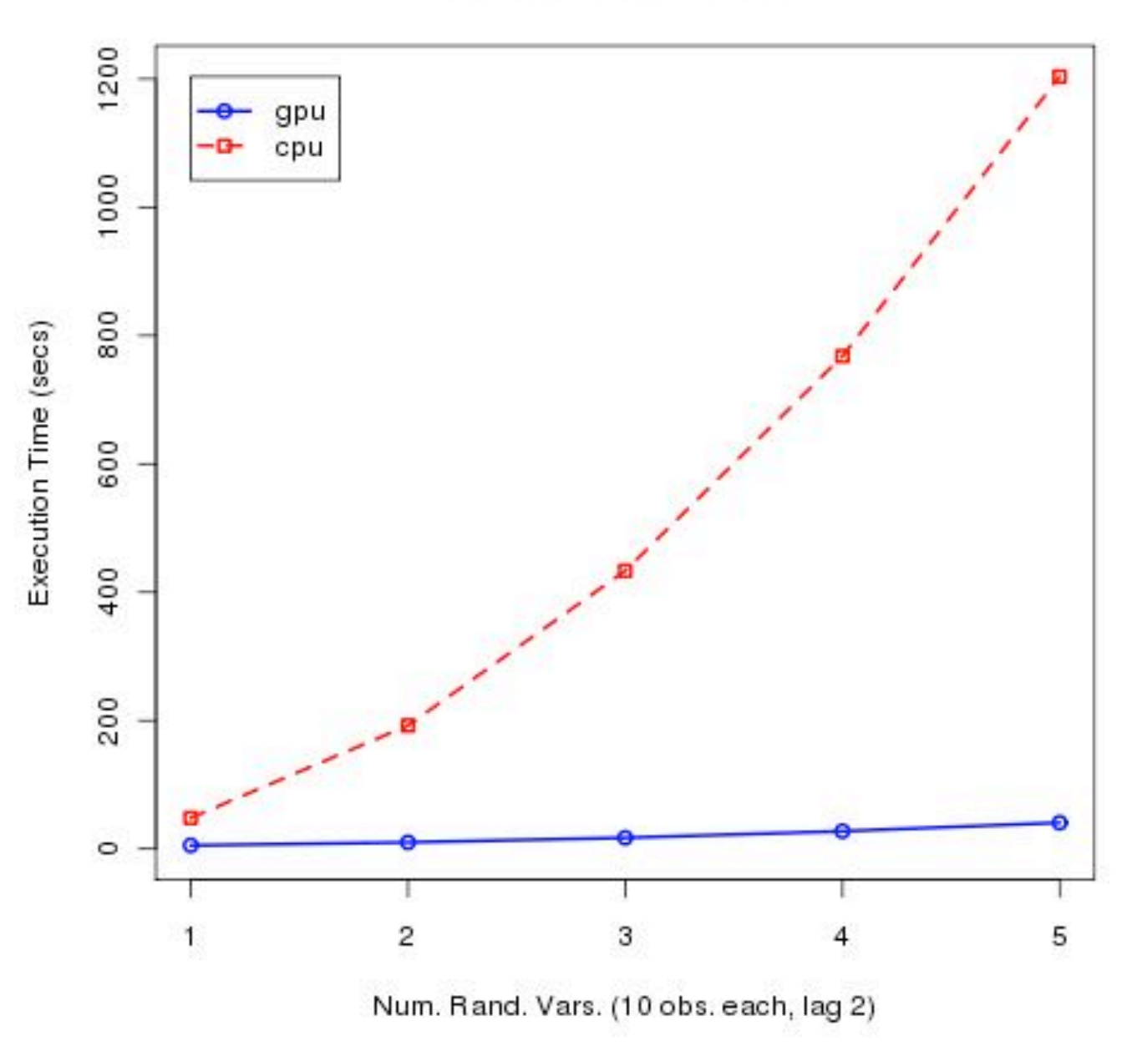

Figure 2 illustrates a performance comparison between a function similar to the 'granger.test' from the package 'MSBVAR' and our gpuGranger function. We use a 4 threads on Intel Core i7 920 versus a GTX 295 GPU.

Fig. 3: Distance & Cluster Times

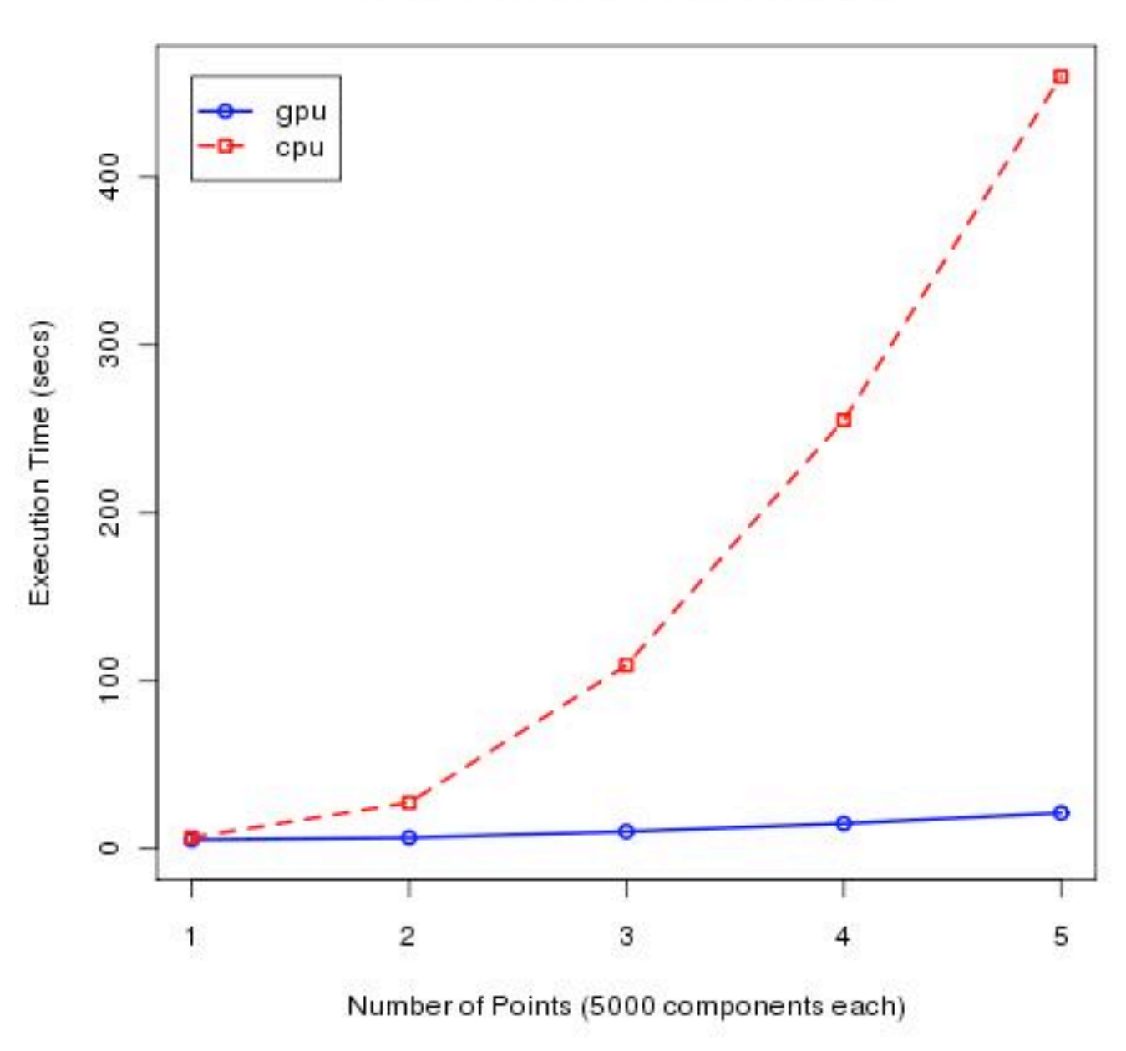

Figure 3 illustrates a performance comparison between the R function 'hclust' composed with 'dist' and our gpuDistHclust function. We use 4 threads on Intel Core i7 920 versus a GTX 295 GPU. Euclidean distance and complete hierarchical clustering was tested with 1000, 2000, 4000, 6000, and 8000 points where each point has 5000 components.

Fig. 4: Kendall Times

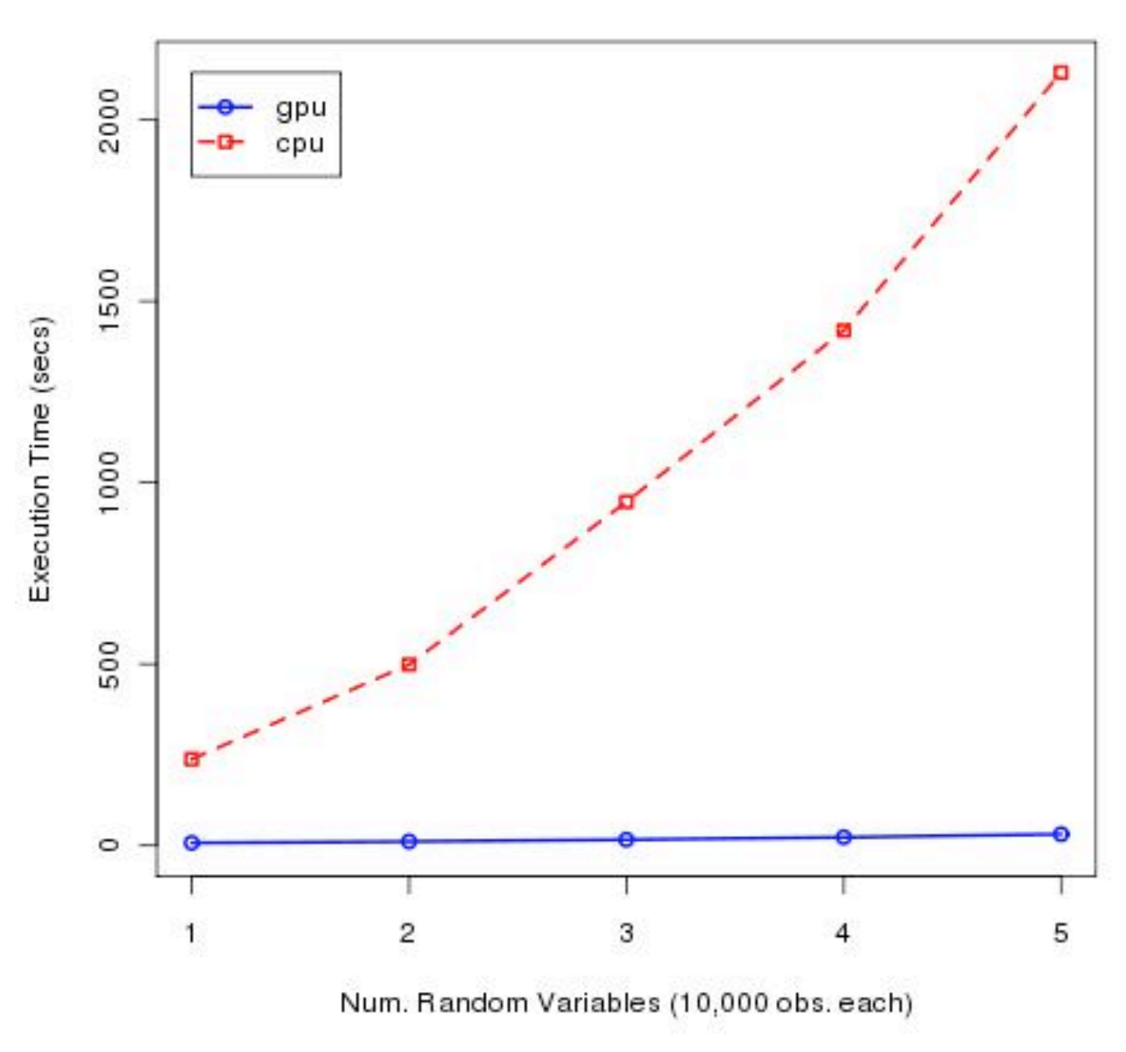

Figure 4 illustrates a performance comparison between the function 'cor' and our gpuCor function with 'method' set to 'kendall'. We use 4 threads on Intel Core i7 920 versus a GTX 295 GPU. Calculation of Kendall's correlation coefficient was tested with 20, 30, 40, 50, and 60 random variables with 10000 observations each.

Acknowledgements

Acknowledgements: J. Buckner, M. Dai, F. Meng, and S. J. Watson 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 partly supported by the National Center for Integrated Biomedical Informatics through NIH grant 1U54DA021519-01A1 to the University of Michigan.

Project website: http://brainarray.mbni.med.umich.edu/brainarray/rgpgpu

Software available at http://cran.r-project.org/web/packages/gputools/index.html