A Heterogeneous Accelerator Platform for Multi-Subject Voxel-based Brain Network Analysis

(Invited Paper)

Yu WANG*, Mo XU*, Ling REN*, Xiaorui ZHANG*, Di WU*, Yong HE†, Ningyi XU‡, Huazhong YANG*
*Department of Electronic Engineering
Tsinghua National Laboratory for Information Science and Technology Tsinghua University
Email: yu-wang@tsinghua.edu.cn
† State Key Laboratory of Cognitive Neuroscience and Learning, Beijing Normal University
‡Hardware Computing Group, Microsoft Research Asia

***Abstract*—The research on understanding the human brain has attracted more and more attention. A promising method is to model the brain as a network based on modern imaging technologies and then to apply graph theory algorithms for analysis. In this work, we examine the computing bottleneck of this method, and propose a CPU-GPU heterogeneous platform to accelerate the process. We construct a statistical brain network from a sample of 198 people and get characteristics such as nodal degree and modularity. This is the first study of voxel-based brain networks on large samples. We also illustrate that domain-specific hardware platform can have a significant impact on neuroscience studies.**

***Keywords*-Heterogeneous Platform; GPU Acceleration; Human Connectome; Voxel-based Brain Network Analysis**

I. Introduction

The Human Connectome is a comprehensive structural description of the connectivity of the human brain [1]. It has attracted great research attention. A promising method is to model the human brain as a network based on BOLD signals (Blood Oxygen Level Dependent) acquired from fMRI (functional Magnetic Resonance Imaging) and then to employ graph theory algorithms to analyze the network [2]. This method is known as Brain Network Analysis (BNA). A different method models the biological neural systems as Spiking Neural Networks (SNN) [3]. This method is based on modeling abstraction rather than imaging technologies. We focus on BNA in this paper.

There are typically two methods with different granularity for BNA: the region-based method and the voxel-based method. A typical region-based study divides the human brain into about 100 regions according to transcendental anatomical knowledge [4]. These studies come up with some important results, such as the scale-free and small-world property of the human brain [5]. However, the results are still inconclusive because the region-based method 1) does not take into account the intra-regional connectivity and functional interactions [4]; 2) may be biased since it relies on predefined anatomical structures [6].

In contrast, a brain network on the voxel-scale is a finer-grained representation of the human brain, usually containing 20K to 100K nodes. It can take advantage of the improving resolution of the imaging technologies and provide more insights into the human brain. Some recent studies adopt the voxel-based method [4], [7].

However, voxel-based BNA encounters two limitations. One is the greater computational complexity as a result of higher resolution. Many graph algorithms become intolerably time-consuming when the network gets too large. Such algorithms include modularity detection and All-Pairs Shortest Path (APSP).

The other limitation is the inherent low signal-to-noise ratio (SNR) of individual voxel data [8]. An easy and effective way to improve SNR is to average a large number of networks from different subjects. In previous voxel-based analysis, however, the amount of subjects is relatively small (less than 30) [4], [9], mainly because the construction of voxel-based networks costs huge time with traditional softwares. The limited amount of samples makes the network over-sensitive to noise and downgrades the reliability of the results.

From the above analysis, it's obvious that a platform with strong computational capability will greatly benefit the voxel-based BNA research. So in this work, we propose a heterogeneous platform consisting of multi-core CPU and GPU (Graphic Processing Unit) to accelerate the process. In our platform, Correlation calculation, Modular Detection and APSP are accelerated on the GPU. Other procedures are implemented on multi-core CPU. Our platform enables researchers to utilize fMRI data with higher resolution, to include more subjects to reduce noise, and to apply precise algorithm for higher accuracy. More importantly, the overall computing time is greatly reduced compared with traditional serial programs. With this platform, we construct and analyze the statistical voxel-based brain networks (consisting of 40k voxels) constructed from 198 people. The whole process finishes in less than 10 hours.

The rest of the paper is organized as follows. We describe our platform in detail in Section II. Then the performance data and the biological results are provided in Section III. Finally, we discuss and conclude our work in Section IV.

978-1-4577-1400-9/11/$26.00 ©2011 IEEE

II. Platform for Multi-subject voxel-based BNA

A. Platform Overview

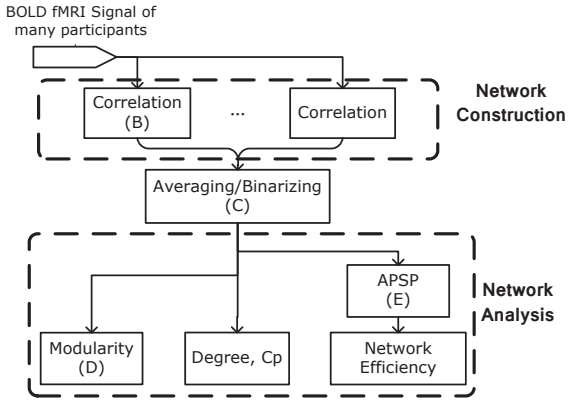

Fig. 1. Work flow of our Platform for multi-subject voxel-based BNA

Fig. 1 is the work flow of our platform. The flowchart can be divided into two parts: network construction and network analysis. In the first part, Pearson Correlation is first calculated to obtain each subject's correlation matrix. A correlation matrix is symmetric, with its element $r_{ij} \in [-1, 1]$ reflecting the correlation level between voxel i and j. These correlation matrices are then averaged and "binarized" into one single adjacency matrix, elements of which are Boolean. If the amount of subjects is large enough, the averaged adjacency matrix can be regarded as the "representative" brain network (or the statistical network). In the network analysis part, network characteristics such as clustering coefficient (Cp) are obtained for the representative brain network. Some of them depend on the results of APSP.

We release our accelerator toolbox on-line (http://parabna.weebly.com) so that researchers in the neuroscience field can benefit from our work. Though we construct brain networks with fMRI data as an example, any other application that involves large graph analysis can also take advantage of the network analysis part of our platform (the bottom block in Fig. 1).

B. Correlation Calculation on GPU

The first step in the work flow is the correlation calculation. The input is BOLD signals of length L for each of the N_v voxels. The Pearson's correlation [10] between node (v_i, v_j) is defined as follows:

$$\hat{r}_{i,j} = \frac{\sum (v_i - \bar{v_i})(v_j - \bar{v_j})}{\sqrt{\sum (v_i - \bar{v_i})^2 \sum (v_j - \bar{v_j})^2}} \quad (1)$$

$$= \frac{\sum v_i v_j - L\bar{v_i}\bar{v_j}}{\sqrt{(\sum v_i^2 - L\bar{v_i}^2)(\sum v_j^2 - L\bar{v_j}^2)}} \quad (2)$$

where v_i denotes the BOLD series of voxel i, $\bar{v_i}$ is the average of the series of that voxel, L is the length of time serials, and all $\sum$ sums along the whole time series $\sum_1^L$.

The Pearson's correlation of one voxel-pair does not rely on other voxel-pairs. So it's suitable to be accelerated on GPU. In our implementation, the first and second moments, i.e $\bar{v_i}$ and $\sum v_i^2$, are calculated first and stay in GPU memory during the kernel execution. Data series of one voxel are consecutively stored to ensure the coalesced memory access. We use 64 threads (a wavefront in AMD GPU) for one voxel-pair correlation. Detailed implementation can be found in our previous work [11].

C. Averaging and Binarizing

In this step, we simply average the correlation matrices and then set a threshold to binarize the mean correlation matrix into an unweighed, undirected graph. We adopt the Ping Pong operation to achieve the best data transfer rate. In the Ping Pong operation, there are two threads working simultaneously, one for reading data from discs, the other for calculating and writing data back. The two threads exchange their tasks in every cycle. In this way, the computation partly overlaps with hard disk reading.

D. Modular Detection on GPU

Modular detection identifies the functionally associated components of the brain. Most previous studies used approximate algorithms, such as the random-walk method [12], [7] and greedy algorithm [13], [2], to reduce the computational complexity. The algorithm we choose is the eigenvector-based spectral partition method proposed by Newman [14]. This algorithm is more precise but at the same time involves much more computation than the above approximate algorithms. Our GPU implementation greatly accelerates this algorithm and makes it applicable to very large graphs.

The idea of Newman's algorithm is to find groups of points that has a lot of inner-group connections and few inter-group connections. A benefit function Q is introduced to judge the network's modularity,

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - P_{ij}] \, \delta(g_i, g_j) \quad (3)$$

where A_{ij} is the adjacency matrix, P_{ij} is the probability for an edge to fall between vertex i and j, g_i indicates to which group voxel i belongs, and $\delta(g_i, g_j)$ is 1 if $g_i = g_j$ and 0 if otherwise. Then the problem becomes finding the best division that maximizes Q. In [14], it is proven that the best division can be obtained by the eigenvector of the most positive eigenvalue of a Modularity Matrix $\mathbf{B} = \mathbf{A} - \mathbf{P}$. Hence we can divide the network into two groups according to the signs of the elements of this eigenvector. The brain networks are unlikely to have only two communities. A modified algorithm to handle multiple division is also described in [14].

This modularity algorithm is based on iterations and is hence fundamentally serial. So it is difficult to migrate the entire algorithm to GPU. Instead, we implement a finer granularity acceleration of the algorithm. In each round of iteration, some basic linear algebra operations, such as sparse matrix multiplication and vector addition and multiplication,

account for most of the computing time. These operations are of high parallelism and very suitable for GPU implementation. So we construct a GPU runtime library of these linear algebra algorithms and use CPU to perform the general scheduling. A detailed description can be found in our previous work [11], which is the first GPU implementation of this algorithm.

E. All-Pair Shortest Paths

APSP algorithms for graphs with diverse characteristics have been studied in depth. Johnson designed an efficient algorithm for sparse graphs [15] based on single-source shortest path such as Dijkstra algorithm [16] and Bellman-Ford algorithm [17]. In addition, when applying to unweighed graphs (this is the case of BNA), Johnson's algorithm reduces to Breadth-First Search (BFS). BFS is very efficient with sparse graphs but performs poorly with dense graphs. Unlike BFS, Floyd-Warshall's algorithm (FW) [18], [19] has O(V^3) time complexity, which is irrelevant to the graph sparsity. Blocked FW algorithm [20] is an improved version of FW algorithm. Not only is it more efficient than the basic FW algorithm, it is also more suitable for GPU implementation.

In BNA, networks with different sparsity are studied, so both the algorithms are useful. We provide both the algorithms in our toolbox so that users can make the optimal choice according to the network sparsity in their application. BFS algorithm is unsuitable for GPU implementation, since it requires the storage of the entire graph, which usually exceeds the GPU memory limit. So we implement it on multi-core CPU. For Blocked FW algorithm, we make some further optimization based on [21]. We also present a performance model for Blocked FW algorithm, which can help decide the optimal block size.

1) BFS on CPU: The implementation is straightforward. All threads traverse across all the graph vertices as source points. Each thread is in charge of its proportion of source nodes, and finds the adjacent but unmet vertices iteratively according to the adjacency matrix. It is worth noting that sorting the vertices according to their degree benefits the load balance of discrete threads.

2) Blocked FW on GPU: We first describe Blocked FW algorithm briefly, and then introduce our further optimization.

The algorithm is not very intuitive. The whole adjacency matrix is first converted to the cost matrix $\mathbf{C}$, where each element C_{ij} represents the path-length of a voxel-pair (i, j). Then the cost matrix is divided into r sub-blocks of equal size [20]. The outer loop iterates over the r primary blocks (the blocks along the diagonal of the matrix). In each of the r iterations, all blocks are updated in the similar way with FW algorithm [18], [19], specifically

$$C_{ij} = min\left(C_{ij}, C_{ik} + C_{kj}\right) \tag{4}$$

where element (k, k) is in the primary block. Therefore, in each iteration, updating a block needs 1) the block in the same column with itself and in the same row with the primary block, denoted with vertical lines in Fig.2, and 2) the block in the same row with itself and in the same column with the primary block, denoted with horizontal lines. According to different memory requirement, one single iteration can be divided into three phases that are performed sequentially (shown in Fig.2). In Phase 1, the primary block itself is updated. In Phase 2, all blocks sharing the same row or column with the primary block are updated. In the last phase, the remaining blocks are updated. In each phase, shadowed block(s) is(are) updated. For example, updating the block with a round dot needs the blocks with vertical or/and horizontal lines. Due to the symmetry of the APSP result, in each iteration, the block updating is performed $r * (r + 1)/2$ times.

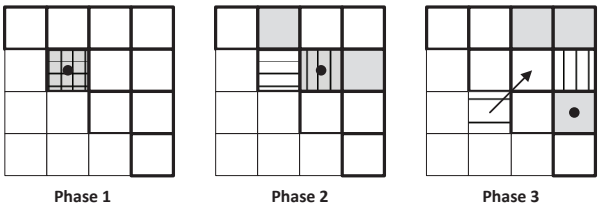

Fig. 2. Illustration of Blocked FW algorithm

In our GPU implementation, block dispatching and phase scheduling are performed by CPU. All blocks are updated serially. In order to minimize the data transfer from the host CPU to the GPU device, the blocks updated in phase 1 and phase 2 are stored in the GPU on-board memory for the coming calculation in phase 3. Therefore the maximum memory usage has the size of r blocks, where r equals to the number of diagonal blocks. By choosing proper block size, our implementation is suitable for networks of any size.

We also adopt several common optimization strategies such as ensuring the coalesced global memory accesses, avoiding shared memory bank conflicts, adjusting the size of the thread-block to fully utilize the computing units, and so on. Besides, since our adjacency matrix is symmetric and we only store an upper block triangular matrix in the host memory, we must transpose the blocks which are not stored. By using a separate kernel for matrix transposing to ensure the coalesced access, our performance is further improved.

3) Performance model of Blocked FW algorithm: The total running time T_{total} only depends on the network size N and the number of blocks $r \times r$. So we propose a performance model: $T_{total}(N, r)$ is a function of N and r. Let n be the dimension of a block. The time complexity of FW algorithm is O(n^3) and the processing of each block is identical. Therefore we can assume the computing time of one block is

$$t_{block} = an^3 + bn^2 + cn + d \tag{5}$$

where $n = \frac{N}{r}$. Hence the total running time is

$$T_{total} = (an^3 + bn^2 + cn + d)\frac{r^2(r+1)}{2} \tag{6}$$

$$= (aN^3 + bN^2r + cNr^2 + dr^3)\frac{r+1}{2r} \tag{7}$$

From Equation (7), T_{total} has a fraction of $\frac{1}{r}$. So when the network size N is fixed, the total time first decreases then

increases as r increases. Therefore an optimal block number exists for a given network. The optimal r_{opt} can be calculated with the following equation:

$$\frac{\partial T_{total}}{\partial r}|_{r=r_{opt}} = 0 \quad (8)$$

In practice, by recording the computing time of networks with different sizes we can obtain the actual model through regression analysis. Then the optimal block number can be calculated.

F. Other network characteristics

The output of our platform is various characteristics of a network. Technically, the modular structure is also an output characteristic, but since its calculation and GPU implementation are complicated, we have discussed it in a separate subsection. Other characteristics include nodal degree, degree distribution, clustering coefficient (Cp), and global/local efficiency. The definition of these characteristics can be found in many literature [6].

Among them, the two most time-consuming procedures are global efficiency and local efficiency. Global efficiency depends on the APSP results of the whole brain network. Local efficiency costs more time; it requires APSP results of many sub-graphs. Here a careful analysis of APSP performance of different algorithms becomes important. If we can choose, at the runtime, the optimal algorithm, and the optimal block size, the calculation of these two characteristics can be further accelerated.

III. Experimental Results

A. Experiment setup

In our experiments, CPU algorithms are implemented with C++ on an AMD Phenom(tm) II X4 965 quad-core CPU @ 3.4GHz and GPU algorithms are implemented with OpenCL, AMD APP SDK v2.4, on an AMD Radeon 5870 GPU. The operating system is Ubuntu 10.04. The dataset was downloaded from a fMRI data sharing project, 1000 Functional Connectomes Project (http://www.nitrc.org/projects/fcon_1000). After the data is preprocessed (including noise reduction), a gray matter mask containing 38368 voxels was applied to all the 198 BOLD signals. The signals are at the spatial resolution of 3mm $\times$ 3mm $\times$ 3mm.

B. Performance

The running time for each part of the processing flow is below.

TABLE I
Performance of Correlation Calculation (one subject)

CPU time(s)	GPU time(s)	Speedup
970	35	27.7

As shown in Table I, we achieve 27.7$\times$ speedup for network correlation on GPU platform over the traditional CPU platform. Note that this procedure has to be preformed to 198 subjects, which means we reduced the total computing time from 2.4 days to 2 hours. In our experiment, the time series length L=225 and the number of voxels N_v=38368 so the thoughput we achieve is $\frac{2LN_v(N_v-1)*4B}{2*35s}$ = 35.3GB/s. The 154GB/s peak bandwidth of GPU global memory is not attained because the reduction operation wastes some of the hardware resources.

TABLE II
Performance of APSP

Sparsity (%)	CPU FW time(s)	GPU FW time(s)	Speedup	multicore CPU BFS time(s)
0.06	416491	2510	165.9	108
0.13	416678	2506	166.3	205
0.39	416829	2519	165.5	528
1.38	415994	2508	165.9	1753
5.46	415822	2499	110.5	6730

Table II shows in detail the performance of APSP. As expected, the running time of FW algorithm stays the same when the sparsity of the graph varies while that of BFS Algorithm is proportional to the sparsity. The GPU Blocked FW algorithm outperforms quad-core CPU BFS algorithm when the sparsity is greater than 2% (due to the curve fit). Our GPU implementation of Blocked FW algorithm is 166$\times$ faster than the CPU version. APSP is an memory bound application. The calculation of the throughput is as follows. From Equation (4), updating an element involves 4 accesses: 3 fetches and 1 write (4*4 Bytes). Each element in the cost matrix is updated N_v times, and there are $N_v(N_v-1)/2$ elements. So the throughput can be calculated $\frac{16N_v^2(N_v-1)}{2T_{total}}$=166GB/s. This throughput is greater than the GPU global memory peak bandwidth 154GB/s because of the data reuse achieved by using shared memory.

TABLE III
Performance of Modular Detection

Sparsity (%)	CPU time(s)	GPU time(s)	Speedup	# of modules
0.06	2954	459	10.7	63
0.13	947	187	12.0	25
0.39	2990	473	17.9	36
1.38	5057	666	26.5	26
5.46	16690	1346	43.6	20

Performance of the Modular Detection is shown in Table III. The running time is positively correlated to the number of parts divided. Our GPU implementation achieves $10 \sim 43\times$ speedups over the single-core CPU version. The speedups are smaller than those in our previous work [11] for the following two reasons. First, we use double-precision in this work (rather than single-precision in [11]) for higher accuracy. GPU suffers more performance losses from double-precision operations than CPU does. Second, the networks in this work is sparser than those used in [11]. From Table III, we can see the speedup is less significant on sparser graphs, where the GPU computing resources are not fully utilized.

C. Validation of the Blocked FW performance model

Fig.3 validates our performance model for Blocked FW algorithm. The model gives accurate prediction most of the

time. But when the block number is large, an observable error occurs between the predictive and the actual results. A possible reason is that when the block number is large, the block size is small and this results in greater relative error. Fig.3 also shows that as the network becomes larger, the error between the theoretical and the experimental results becomes smaller.

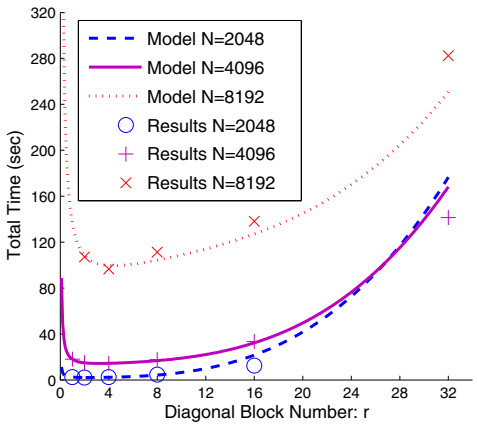

Fig. 3. Total time of the GPU-based Blocked FW algorithm with different network dimensions and diagonal block numbers. In the figure, lines represent the performance model and dots are practical results of our implementation.

For our voxel-based brain networks, the optimal block number r_{opt} is around 12.

D. Biological results

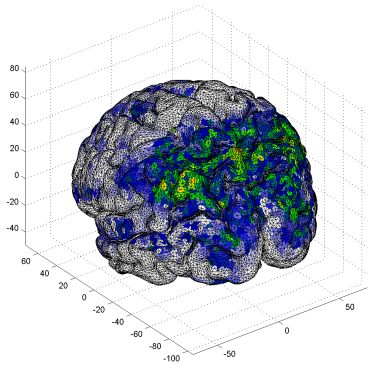

Fig. 4. Nodal degree visualization, Sparsity = 0.38

1) Nodal Degree and Degree Distribution: The nodal degree and the degree distribution are both well-accepted characteristics of the human brain network. In a graph, the degree of a node is defined as the number of edges connected to it, reflecting the connectivity of one voxel to other voxels. Voxels with a degree much higher than the average are considered to be potential hub-voxels [4]. The cerebral cortex is commonly believed to be the most activated area of the whole brain and. So the degree of voxels near cerebral cortex is of essential value for future research and also for evaluating the correctness of our results. Fig. 4 demonstrates the degree of voxels near the cerebral cortex (darker color means higher degree).

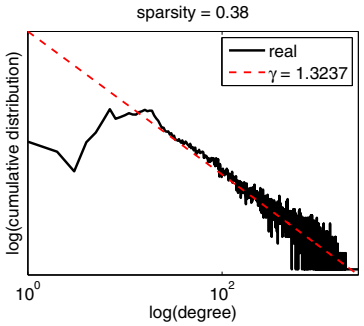

Fig. 5. Degree Distribution, Sparsity = 0.38

The degree distribution $P(k)$ is defined as the fraction of nodes with degree k. Fig. 5 clearly shows that it follows the power law scaling decaying as $P = k^{-\gamma}$, where γ is around 2. This power law indicates the scale-free property of brain networks: though most of the voxels are connected to a small number of other voxels, there are a few voxels that have much more connections and are regard as potential hubs.

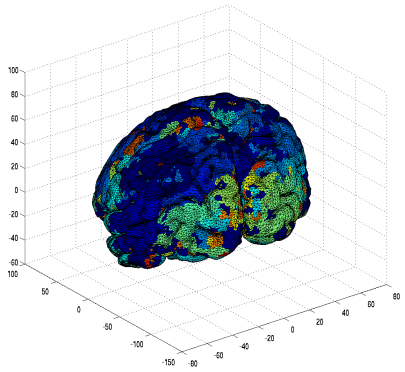

Fig. 6. Modular Detection visualization, sparsity = 0.38

2) Modular Detection: Here we visualize the actual position of each group with different colors in Fig. 6. We can see that neighboring voxels are more likely to be divided into the same module. The number of modules after partition is already provided in Table III. The number of modules are stable in networks with different sparsity, except for the network with sparsity 0.06%. This may be due to the noise or may suggest that network with sparsity 0.06% is not a good representation of the brain.

IV. Conclusion and Discussion

In this work, we propose a heterogeneous accelerator platform for multi-subject voxel-based BNA. This platform takes advantage of commodity hardwares (CPU and GPU), and

greatly enhances the efficiency and accuracy of the brain network analysis.

A. Heterogeneity

We accelerate three most time-consuming processes on GPU, i.e. Correlation in the network construction, Block FW algorithm for APSP and Modular Detection in the network analysis. These processes share the common feature of high parallelism combined with relatively low host-device data transfer requirements. As we know, data transfer between host CPU and GPU devices is through PCI-e bus. Since the bandwidth of PCI-e bus often limits the overall speedup, it should not be ignored. In these three processes, time complexity of computation is much larger than that of data transfer. That is one prerequisite for achieving satisfactory speedup.

We also show some processes are not suitable for GPU acceleration, such as averaging the adjacency matrices, BFS algorithm for sparse graph and the calculation of some network characteristics. For better integrity, we implement them on multi-core CPU. These processes either are data-transfer intensive or require frequent communication between parallel threads.

Therefore, in our platform, heterogeneous hardware collaborates well in multi-levels. To be specific, first, different processes in the platform are accelerated on different hardwares. Second, to achieve the optimal performance of APSP calculation under different sparsity, CPU and GPU implementation are both well-prepared. Third, each GPU program itself needs the help of CPU for scheduling and data dispatching.

B. Applicability, scalability and reusability

The main advantages of our platform over other hardware platforms such as clusters is that ours has smaller size, consumes lower power and costs less. It means that neuroscience researchers can benefit from our platform without much investment. It is also possible to integrate our platform with future fMRI machines.

Apart from integrity, our platform also enjoys necessary scalability and reusability. Since most of the accelerations on GPU are matrix operations, and the operations are ready to be divided into blocks, it is easy to adjust the implementation for larger networks analysis.

Since our platform is made up of independent functions, these functions can be applied to other applications, even in other fields. In the neuroscience research, other methods to explore the brain may also need the calculation of network characteristics.

In the future, we will keep consummating our platform and integrating other functions. There is a myriad of Brain Network Analysis methods that need to be accelerated. We are going to form a long-term cooperation with Brain Network researchers and keep on the work of acceleration.

V. ACKNOWLEDGEMENT

This work is supported by Microsoft Research Asia and AMD China University Program. This work is also partially supported by National Natural Science Foundation of China (No.60870001), 863 project (No. 2009AA01Z130).

REFERENCES

[1] O. Sporns, G. Tononi, and R. Ktter, "The human connectome: A structural description of the human brain," *PLoS Comput Biol*, vol. 1, no. 4, p. e42, 09 2005.

[2] Y. He, J. Wang, L. Wang, Z. J. Chen, C. Yan, H. Yang, H. Tang, C. Zhu, Q. Gong, Y. Zang, and A. C. Evans, "Uncovering intrinsic modular organization of spontaneous brain activity in humans," *PLoS ONE*, vol. 4, no. 4, p. e5226, 04 2009.

[3] J. L. Krichmar, N. Dutt, J. M. Nageswaran, and M. Richert, "Neuromorphic modeling abstractions and simulation of large-scale cortical networks," in *ICCAD*, 2011, pp. 1–5.

[4] M. van den Heuvel, C. Stam, M. Boersma, and H. Hulshoffpol, "Small-world and scale-free organization of voxel-based resting-state functional connectivity in the human brain," *NeuroImage*, vol. 43, no. 3, pp. 528–539, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.neuroimage.2008.08.010

[5] V. M. Eguíluz, D. R. Chialvo, G. A. Cecchi, M. Baliki, and V. A. Apkarian, "Scale-Free brain functional networks," *Physical Review Letters*, vol. 94, no. 1, pp. 018 102+, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.94.018102

[6] J. Wang, X. Zuo, and Y. He, "Graph-based network analysis of resting-state functional MRI." *Frontiers in systems neuroscience*, vol. 4, 2010. [Online]. Available: http://dx.doi.org/10.3389/fnsys.2010.00016

[7] M. Valencia, M. A. Pastor, M. A. Fernández-Seara, J. Artieda, J. Martinerie, and M. Chavez, "Complex modular structure of large-scale brain networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 2, p. 023119, 2009.

[8] R. Heller, D. Stanley, D. Yekutieli, N. Rubin, and Y. Benjamini, "Cluster-based analysis of FMRI data." *Neuroimage*, vol. 33, no. 2, pp. 599–608, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.neuroimage.2006.04.233

[9] M. P. van den Heuvel, C. J. Stam, R. S. Kahn, and Hulshoff, "Efficiency of functional brain networks and intellectual performance," *J. Neurosci.*, vol. 29, no. 23, pp. 7619–7624, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1523/JNEUROSCI.1443-09.2009

[10] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[11] D. Wu, T. Wu, Y. Shan, Y. Wang, Y. He, N. Xu, and H. Yang, "Making human connectome faster: Gpu acceleration of brain network analysis," in *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ser. ICPADS '10. Shanghai, China: IEEE Computer Society, 2010, pp. 593–600. [Online]. Available: http://dx.doi.org/10.1109/ICPADS.2010.105

[12] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006.

[13] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, no. 6, p. 066133, Jun 2004.

[14] ——, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E*, vol. 74, no. 3, p. 036104, Sep 2006.

[15] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271–271, December 1959.

[17] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[18] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

[19] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[20] G. Venkataraman, S. Sahni, and S. Mukhopadhyaya, "A blocked all-pairs shortest-paths algorithm," in *Lecture Notes in Computer Science*, 2000.

[21] G. J. Katz and J. T. Kider, Jr, "All-pairs shortest-paths for large graphs on the gpu," in *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 47–55.