A GPU-accelerated Framework for Fast Mapping of Dense Functional Connectomes

Kang Zhao, Haixiao Du and Yu Wang
Department of Electronic Engineering, Tsinghua University
Beijing, China
Email: {zhaok14,duhx11}@mails.tsinghua.edu.cn,yu-wang@tsinghua.edu.cn

Abstract—**In the context of voxel-based modalities like functional magnetic resonance imaging (fMRI), a dense connectome can be treated as a large-scale network where single voxels are directly used to define brain network nodes. Contrary to parcellated connectomes, dense connectomes have higher spatial resolution and are immune from the parcellation quality. However, the analysis of dense connectomes basically requires more powerful computing and storage capacities. Here, we proposed a graphics processing unit(GPU)-accelerated framework to perform fast mapping of dense functional connectomes. Specifically, the framework is scalable to high voxel-resolution imaging data(<2mm) and can construct large-scale functional brain networks with lower time and memory overheads. Based on the proposed framework, three functional connectivity measures (Pearson's, Spearman's and Kendall's) were accelerated on the GPU for fast detection of possible functional links in dense connectomes. Experimental results demonstrated that our GPU acceleration for the Kendall's measure delivered a >50x speedup against both multi-core CPUs implementations and GPU-based related works.**

Keywords–neuroinformatics; dense connectomes; functional connectivity measures; GPU; voxel resolution.

I. INTRODUCTION

Recent advances in resting-state functional magnetic resonance imaging (rs-fMRI) technologies have provided a non-invasive way to depict spontaneous fluctuations in brain activity and thus facilitates the mapping of functional connectomes [1]. Thanks to the constant increase of imaging resolution, researchers nowadays are able to analyze functional connectivity patterns of human brain at a finer spatial resolution, which triggers the rise of 'dense connectome' study[2][3].

A dense functional connectome is generally modeled as a large-scale network whose nodes can be defined directly by voxels in fMRI imaging data [4]. The investigation of voxel-wise functional networks allows to uncover more detailed connectivity information but is typically coupled with considerable computation and storage demands [5]. Specifically, the total amount of voxels grows cubically as a function of voxel-resolution, leading to a sharp increase in computations when measuring the functional connectivity between all pairs of voxels. Moreover, formally represented by a connectivity matrix, a voxel-wise network requires quadratic complexity of storage with the growth of voxel amounts, which implies a considerable memory footprint for the construction of large-scale functional networks. As shown in Figure 1, at the 1mm resolution of approximate 1,600,000 voxels, more than 8 TB memory is required for the storage of the voxel-wise whole-brain connectivity matrix. Taken together, the computation and storage requirements are the most pressing problems in the study of dense functional connetomes.

Given the limited computational power, extensive research has attempted to scale down brain networks by either down-sampling the imaging data towards a coarser level, or aggregating network nodes to several large parcels in the light of anatomically or functionally-defined brain atlases, i.e., parcellated connectomes [6]. However, it is quite obvious that these solutions may lead to the loss of potentially significant connectivity information, not to mention that the analysis of parcellated connectomes are highly sensitive to the parcellation selection [7].

In recent years, the advent of general-purpose graphics processing units (GPGPUs) opens a new door to gigantic data processing [8]. Benefiting from many-core architectures, GPUs exhibit a high bandwidth and tremendous computational horsepower, and the collaboration of CPU-GPU can achieve remarkable performance boosts for many applications [9]. In the field of imaging connectomics, several attempts have been made to accelerate the mapping of dense connectomes using GPUs [10][11]. Nonetheless, these studies either are powerless in the treatment of high resolution data (e.g., 2mm or higher resolutions), or present a rapid deterioration of performance as the growth of voxel aggregates. Scalable GPU-based algorithms used to map dense functional connectomes are currently lacking.

In this paper, we proposed a GPU-accelerated framework aimed for fast mapping of dense functional connectomes. Specifically, the proposed framework enables fast construction of large-scale functional network based on three distinct functional connectivity (FC) measures: Pearson's, Spearman's and Kendall's measures [12]. Moreover, attributed to a novel memory optimization strategy, our framework is scalable to the high-resolution imaging data (<2mm). Experimental results showed that running on a single-GPU system, our framework can extract large-scale functional networks (10^6 nodes, 1% edge sparsity) within 1000 seconds.

The remaining part of this paper proceeds as follows. In section II, we begin with an overview on the general flow of functional network construction, focusing on the possible challenges for mapping dense connectomes. After that, a scalable GPU-accelerated framework and the corresponding accelerated methods of FC measures are described in order to tackle these challenges. The performances of these algorithms are analyzed in Section III. Section IV discusses the major contributions of our study along with some future expectations.

II. METHODS AND MATERIALS

This section details three aspects: the basic steps and issues of voxel-wise brain network construction, our solutions under different thresholding and FC measuring approaches as well as the design of experiments including data generation and the selection of benchmarks.

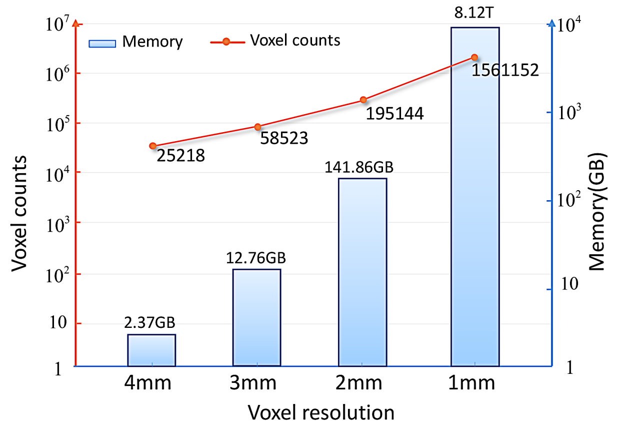

Figure 1. For different imaging resolutions, the corresponding number of voxels, and memory requirements for generating connectivity matrices. Source of statistical data: http://fcon_1000.projects.nitrc.org/indi/CoRR/html/bnu_1.html.

Algorithm 1 Constructing networks given the connectivity strength (CS) threshold

```
Input:data matrix D, the CS threshold CSthreshold;
Output:the resultant network ResultNet_host;
Variables defined on CPU main memory: D_host,
  ResultNet_host;
Variables defined on GPU memory: D_dev, Batch_dev;
 1: Transfer D_host to D_dev;
 2: Partition D_dev into $m$ blocks, numbered from $\mathbf{D}_1$ to $\mathbf{D}_m$;
 3: for $row \leftarrow 1$ to $m$ do
 4:     for $column \leftarrow$row to $m$ do
 5:         Batch_device$\leftarrow$ GPU_$f$ $(\mathbf{D}_{row}, \mathbf{D}_{column})$;
 6:         GPU_thresholding(Batch_dev, CSthreshold);
 7:         GPU_compressing(Batch_dev);
 8:         Transfer Batch_dev to Batch_host;
 9:         CPU_assemble(Batch_host,ResultNet_host);
10:         Batch_dev.clear();
```

A. *An Overview on Functional Network Construction*

In fMRI-based functional connectomics, the imaging data of a certain subject is acquired by successively recording blood oxygenation level-dependent signals at each imaging voxel site [4]. Then, the imaging data is preprocessed by some conventional means (e.g., slice timing correction, spatial and temporal filtering) before it is finally represented by a data matrix $\mathbf{D}^{N\times L}$, where N is the number of voxels and L is the length of time series [13]. After that, the construction flow of a functional network can be typically summarized into two main steps: generating connectivity matrix and thresholding. Firstly, the functional connectivity strength between any two voxels is calculated via diverse FC measures to describe how N distinct voxels functionally interact with each other, which generates an $N \times N$ connectivity matrix. Once a connectivity matrix is generated, given the consensus that human brain functional networks organize as an economical small-world topology tending to minimize wiring costs [14], a subsequent thresholding procedure should be applied to remove spurious connections to ensure the sparsity nature of the brain networks [15]. At present, there is no gold standard for the set of threhold values. Generally a sensitive analysis across diverse thresholds is recommendatory for researchers to seek appropriate thresholding parameters[16]. Finally, the sparse networks established though thresholding procedures can be compressed into a sparse format with lower memory footprints, e.g., the compressed sparse row (CSR) format [17].

For the construction of voxel-level networks, it is noteworthy that despite underlying huge memory demands of intermediate connectivity matrix (Figure 1), the established networks after thresholding are normally less memory-hungry. For example, under 1mm isotropic resolution, the established sparse network with 0.1% edge density requires only $\sim$16 GB memory, much lower than that of the entire connectivity matrix. Hence, the basic idea of our proposed framework is to avoid maintaining the entire connectivity matrix by employing a GPU-based block-wise thresholding strategy.

B. *Scalable Solutions for Voxelwise Network Construction*

As the network size grows dramatically with the increasing voxel aggregates, a scalable method is required for the voxel-wise network construction. Here, we proposed a GPU-based block-wise thresholding strategy under two different thresholding modes: given the connectivity strength threshold or the sparsity threshold [18].

1) Specify Connectivity Strength Thresholds: Assigning the connectivity strength threshold means that a fixed threshold value is set as the baseline when thresholding the connectivity matrix so that matrix elements greater than the given threshold are reserved while others are set to 0s.

In this case, the steps of generating connectivity matrix and thresholding can be easily merged. Specifically, the aforementioned data matrix $\mathbf{D}^{N\times L}$ can be divided into multiple (m) blocks, i.e., $\mathbf{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$, where the block size is adjustable. Then, the corresponding connectivity matrix $\mathbf{R}^{N\times N} = f\left(\mathbf{D}^T, \mathbf{D}\right)$ can be derived by:

$$\mathbf{R} = \begin{bmatrix} f\left(\mathbf{D}_1^T, \mathbf{D}_1\right) & f\left(\mathbf{D}_1^T, \mathbf{D}_2\right) & \ldots & f\left(\mathbf{D}_1^T, \mathbf{D}_m\right) \\ f\left(\mathbf{D}_2^T, \mathbf{D}_1\right) & f\left(\mathbf{D}_2^T, \mathbf{D}_2\right) & \ldots & f\left(\mathbf{D}_2^T, \mathbf{D}_m\right) \\ \vdots & \vdots & \ddots & \vdots \\ f\left(\mathbf{D}_m^T, \mathbf{D}_1\right) & f\left(\mathbf{D}_m^T, \mathbf{D}_2\right) & \ldots & f\left(\mathbf{D}_m^T, \mathbf{D}_m\right) \end{bmatrix}, \tag{1}$$

where f represents distinct FC measures, e.g., the Pearson's measure. In this way, $\mathbf{R}^{N\times N}$ is generated block by block. Once a block is obtained, a subsequent thresholding and compressing procedure is performed immediately instead of doing this after the generation of the entire $\mathbf{R}^{N\times N}$. All the computation, thresholding and compression procedures can be efficiently completed by a GPU device, while the CPU only serves as an assembly line for receiving compressed data from the GPU in series and continuously jointing them together into a complete network that stays in main memory with a sparse format. The execution procedure is shown in Algorithm 1.

It should be noted that during the process, only a sparsely stored matrix is maintained in CPU main memory. Thus, the algorithm has a linear spatial complexity $O\left(N+E\right)$ for storing voxel-level functional networks, where E is the number of valid edges after thresholding. As the network can be quite sparse, the CPU memory usage is largely decreased in this way.

2) Specify Sparsity Thresholds: Another commonly used thresholding strategy is to fix the network sparsity. The sparsity is defined as the proportion of the quantity of existing edges to the maximum possible number of edges in a network. Comparatively, the sparsity-based thresholding approach is more suitable for group-level comparisons on network topology [19] but is more complex to be applied in the construction of large-scale networks because the sparsity threshold should be firstly transferred to a corresponding connectivity strength threshold, which is in need of a time-consuming statistic for all entries in the connectivity matrix $\mathbf{R}^{N\times N}$.

In response, a GPU-accelerated algorithm characteristic of calculating the connectivity matrix $\mathbf{R}^{N\times N}$ for two times was designed. During the first generation, the GPU analyzes the distribution of element values in $\mathbf{R}^{N\times N}$ to derive the connectivity strength threshold corresponding to the given sparsity. Considering that a sparsity threshold restricts the number of actual connections k in a network, our basic idea is to find the k-th maximal element r_k in $\mathbf{R}^{N\times N}$ via GPU statistics, to serve as the connectivity strength threshold. Once r_k is obtained, the algorithm described above (Algorithm 1) can be reused to establish networks, which will also satisfy the constraint of the specified sparsity threshold. Then, the running of Algorithm 1 actually requires computing $\mathbf{R}^{N\times N}$ one more time.

Specifically in the first round, the range of the connectivity strength from 0 to 1 is segmented into multiple bins, with the bin quantity N_{bin}, and the bin width $\varepsilon = 1/N_{bin}$. Each time a block of $\mathbf{R}^{N\times N}$ is generated, a statistical histogram is maintained and updated by counting the quantity of elements in the block falling into different bins. A GPU-based sort-search histogram algorithm is applied to attain a fast and stable performance for large number of bins [20]. After finishing statistics of all blocks, the very bin where r_k is located can be find from the statistical histogram and the eventual outcome $\hat{r}$ is set as the median of this bin. The process is summarized in Algorithm 2. Notably, the error between $\hat{r}$ and r_k can be estimated by :

$$|\hat{r} - r_k| \leq \varepsilon/2 = 1/(2N_{bin}), \tag{2}$$

where the precision can be simply improved by increasing the number of bins, i.e., N_{bin}. In practice, we set $N_{bin} = 10^6$, rendering $\hat{r}$ an extreme approximation to r_k. Taken together, by adopting the block-wise statistical and approximate strategy we avoid maintaining the entire connectivity matrix as a whole, thereby reducing the algorithmic demand for CPU/GPU memory.

C. GPU implementation of three FC Measures

FC measures are used to quantify the strength of functional connections between network nodes. In this section, we will detail our GPU-accelerated algorithms for three commonly used FC measures: Pearson's, Spearman's and Kendall's measures, of which the latter two are considered more robust to outlying observations [21]. To accelerate the calculation of FC measures on a GPU, the basic principle of our proposed algorithms is to transform the computation of these measures into normative operations that GPUs excel in, e.g., vectors or matrices multiplications, both of which that possess high parallelism can be executed very quickly on GPUs.

As mentioned above, an fMRI data set of a single subject can be represented by a data matrix $\mathbf{D}^{N\times L} = (d_i)$, where $1 \leq i \leq N$, and $d_i = (d_{i1}, d_{i2} \ldots, d_{iL})$, i.e., the i-th row of $\mathbf{D}^{N\times L}$, denoting the time series of the i-th voxel, with the sequence length L. Thus $r_{i,j}$, the FC measurements between voxel i and voxel j, can be described as follows:

Algorithm 2 Derive the corresponding connectivity strength threshold from the given sparsity threshold

```
Input:data matrix D, the sparsity threshold Sparsity;
Output:the connectivity strength threshold CSthreshold;
Variables defined on CPU main memory: D_host;
Variables defined on GPU memory: D_dev, Batch_dev,
  histogram_dev;
 1: Define $k \leftarrow N \times N \times$ Sparsity;
 2: Transfer D_host to D_dev;
 3: Partition D_dev into $m$ blocks, from $\mathbf{D}_1$ to $\mathbf{D}_m$;
 4: for row ← 1 to $m$ do
 5:     for column ← row to $m$ do
 6:         Batch_dev ← GPU_f ($\mathbf{D}_{row}$, $\mathbf{D}_{column}$);
 7:         GPU_histogram(Batch_dev, histogram_dev);
 8:         Batch_dev.clear();
 9: Define Position ← GPU_upperBound(histogram_dev, $k$);
10: CSthreshold ← binWidth × Position + binWidth/2.0;
```

$$r_{i,j} = f(d_i, d_j), \tag{3}$$

where $f \in \{f_p, f_s, f_k\}$, representing Pearson's, Spearman's and Kendall's measures, respectively, whose definitions will be specified below.

1) Vectorization for Pearson's and Kendall's Measures: The Pearson measure of temporal correlation between two time-series d_i, d_j is defined by:

$$f_p = \frac{\sum_{k=1}^{L}\left(d_{ik} - \bar{d}_i\right)\left(d_{jk} - \bar{d}_j\right)}{S_{d_i} \cdot S_{d_j}}, \tag{4}$$

where $\bar{d}_i = \left(\sum_{k=1}^{L} d_{ik}\right)\Big/L$ and $S_{d_i} = \sqrt{\sum_{k=1}^{L}\left(d_{ik} - \bar{d}_i\right)^2}$ are the mean and standard deviation of the time series of the i-th voxel respectively. Several studies have suggested that f_p be derived as the product of two normalized vectors [22]:

$$f_p = \sum_{k=1}^{L}\left(\frac{d_{ik} - \bar{d}_i}{S_{d_i}}\right)\left(\frac{d_{jk} - \bar{d}_j}{S_{d_j}}\right) = \vec{p}_i \cdot \vec{p}_j, \tag{5}$$

where $\vec{p}_i$, $\vec{p}_j$ are vectors with the length L and $\vec{p}_{ik} = \left(d_{ik} - \vec{d}_i\right)\Big/S_{d_i}$. Likewise, the formula of the Kendall's measure can be vectorized to:

$$f_k = \sum_{k=1}^{L-1}\sum_{q=k+1}^{L}\left(\frac{sign\left(d_{ik} - d_{iq}\right)}{\sqrt{m - m_i}}\right)\left(\frac{sign\left(d_{jk} - d_{jq}\right)}{\sqrt{m - m_j}}\right) = \vec{z}_i \cdot \vec{z}_j \tag{6}$$

where $\vec{z}_{ik} = sign\left(d_{ik} - d_{iq}\right)/\sqrt{m - m_i}$ and m, m_i are scalars [23]. Note that vectors $\vec{z}_i$ and $\vec{z}_j$ have the length $L\left(L-1\right)/2$.

To obtain all-pairs FC measurements, i.e., the connectivity matrix $\mathbf{R}^{N\times N} = (r_{i,j})$, the $\vec{p}_i\left(\vec{z}_i\right)$ of every voxel should be firstly derived. Then, the subsequent operations of all-pairs $r_{i,j} = \vec{p}_i \cdot \vec{p}_j$ (Pearson's measure), or $r_{i,j} = \vec{z}_i \cdot \vec{z}_j$ (Kendall's measure), can be unified as the matrix-matrix multiplication, which is efficient on GPUs.

2) Accelerating Rank Assignments for Spearman's Measure: The Spearman correlation between two time-series can be calculated by measuring the Pearson correlation of their ranked values of each samples [24], i.e.,

$$f_s(d_i, d_j) = f_p(n_i, n_j), \tag{7}$$

where n_i, n_j are the ranked sequences of d_i, d_j, respectively. That is, by replacing every element in a time-series with its rank, our proposed GPU-based procedure of Pearson's measures described above can be applied directly to accelerate the computation of the Spearman's measure. Hence, the key issue is how to calculate element ranks efficiently on a GPU.

A GPU-based sort-detection algorithm was introduced by Kim et al. (2012) [25] for assigning ranks, yet this approach is powerless in process of tied elements, i.e., multiple identical values in one time-series. Here, we propose a new rank assigning strategy calculating the rank of d_{ik} in the *i*-th time series as follows:

$$rank(d_{ik}) = LessNumber + (1 + EqualNumber)/2, \tag{8}$$

where $LessNumber$ is the amount of elements less than d_{ik} and $EqualNumber$ is the amount of tied elements equal to d_{ik} (including itself). To obtain the rank of an element d_{ik}, a GPU only needs to traverse all elements of a time-series and count the number of elements less than or equal to itself, instead of sorting all temporal samples. This strategy avoids possible branch operations and irregular memory access among multiple threads, thereby easily parallelized by a GPU with single instruction, multiple threads (SIMT) model [26]. Performances of the sort-detection algorithm and our own implementation of the Spearman's measure will be compared later.

D. Application and Example Datasets

Several experiments were conducted to illuminate the advantages of our proposed framework in two aspects: the run time efficiency and the scalability. To comprehensively assess the run time efficiency, two sets of data with respective number of nodes *N*=25218 (approximate to the voxel aggregates of a 4mm isotropic resolution imaging data) and *N*=58523 (approximate to the voxel aggregates of a 3mm isotropic resolution imaging data) were randomly generated and stored using a floating point format of 4 bytes per element. In addition, each data set has four variants with varied number of temporal samples (*L*=128, 256, 512, 1024), to evaluate the performance of our GPU-accelerated algorithms under different length of time series. In contrast, to investigate the scalability of the proposed framework, another input data set was produced with constant length of time series (*L*=128) but spanning a broad range of voxel quantities (*N*= 200,000, 400,000, 600,000, 800,000, 1000,000, respectively).

E. Benchmarks and Programs

All experiments were conducted on a workstation with an Intel(R) Core(TM) i7-6700K CPU (4G Hz, 8 cores, hyper-threading disabled), 64GB main memory, and an NVDIA GeForce GTX TITAN Black GPU with 6 GB device memory. The workstation supports dual operation systems (Windows 8.1 and Linux Ubuntu 16.10). MATLAB(R2016a) and CUDA(v8.0) are available on both systems.

So far, several parallel-processing approaches have been put forward to accelerate the calculation of FC measures. Here we consider two class of typical related works as comparisons: multi-core CPUs based implementations, and GPU based implementations by others.

The contrastive programs of Pearson's and Spearman's measures on multi-core CPUs are parallelized by invoking the Intel Math Kernel library(MKL), a widely used math library featuring highly optimized and easily parallelizable functions on multi-core systems [27]. Besides, the parallel implementation of the Kendall's measure on multi-core CPUs is based on a classical algorithm built upon merge sort [28]. On the current workbench, all these CPU-based programs are parallelized using 8 threads. As for GPU-based related works, the programs from *gputools*, a prevalent toolbox enabling efficient GPU computing in R [29], are picked up for the performance comparison on Pearson's and Kendall's measures, and the aforementioned sort-detection algorithm proposed by Kim et al. (2012) [25] is re-implemented and tested against our own implementation of the Spearman's measure.

III. RESULTS

We first assessed the performance of our GPU-accelerated algorithms for three FC measures (Figure 2). In this case, all programs were required to generate full-stored connectivity matrices without thresholding and compression operations. Then, the elapsed time of constructing sparse networks among different network scales and sparsity thresholds was presented in Table 1 to highlight the scalability of our framework.

A. Performance

Experimental results in Figure 2 showed that our proposed GPU-accelerated algorithms for Pearson's, Spearman's and Kendall's measures were more time-efficient against both multi-core CPUs and GPU based related works. In particular, our own implementation of the Kendall's measure exhibited a over 50x speedup than the other two ways.

Specifically, our GPU implementation of Pearson's measures only cost 1.7 seconds on average for generating the connectivity matrix at a 4mm isotropic resolution (*N*=25218) across different lengths of time series, and 4.3 seconds at a 3mm isotropic resolution (*N*=58523). Similarly, for the Spearman's measure, the average time costs were 2.0 and 4.4 seconds, respectively, at the two network scales. Actually, our GPU-accelerated procedure of the Spearman's measure performed only slightly slower than that of the Pearson's measure. Considering that the execution of Spearman's measure internally called the procedure of the Pearson's measure upon finishing rank assignments, the minor variance in computing time between the two measures implied the high efficiency of our proposed strategy for rank assignments, which led to the little time occupancy of this step during the calculation of the Spearman's measure. As for the Kendall's measure, our procedure spent averagely 63.5 and 340.5 seconds generating the connectivity matrix at the resolution of 4mm and 3mm, respectively, which was much less than that of the multi-core CPUs implementation using MKL parallel computing library [27] or the GPU implementation using *gputools* [29]. The latter two ways both consumed >1 hour dealing with 4mm resolution data and >5 hours with 3mm data. Notably, the calculation of the Kendall's measure that has a quadratic time

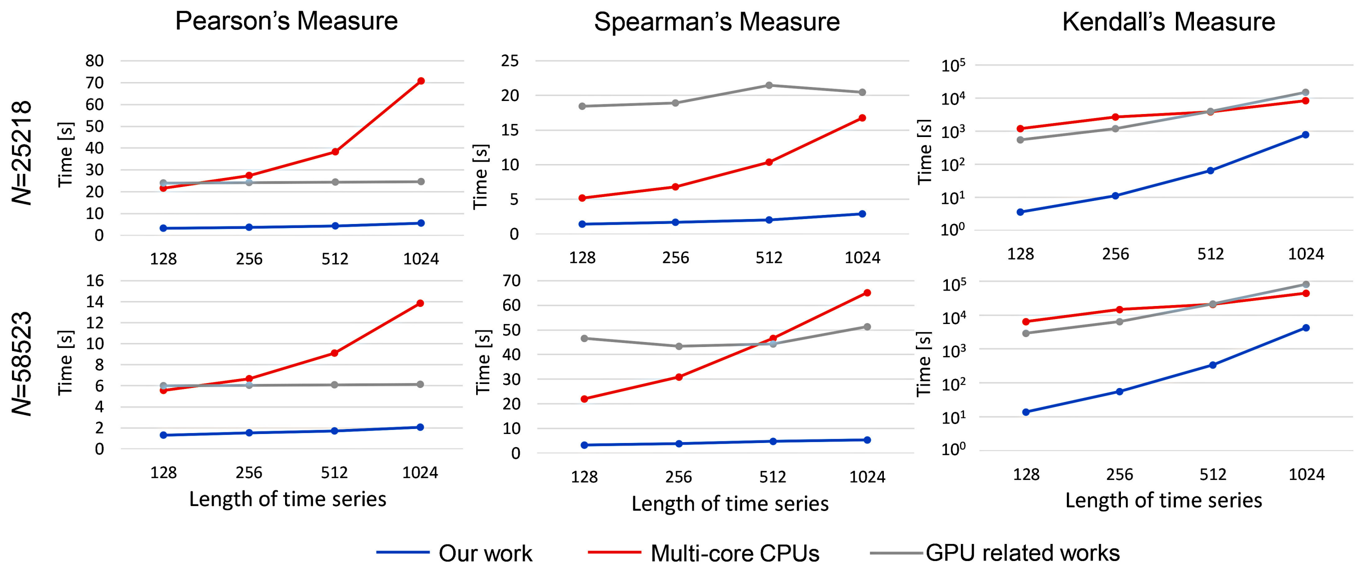

Figure 2. Performance comparisons of different implementations of three FC measures (Pearson's, Spearman's and Kendall's).

complexity as the increase of lengths of time series is generally more time-consuming than that of Pearson's and Spearman's measures with linear span.

Additionally, it was observed that multi-core CPUs implementations of Pearson's and Spearman's measures were adversely sensitive to the length of time series. Moreover, GPU-based related works regarding three measures, due to their lack of specialized treatments to efficiently adapt these measures on the GPU architecture, showed relatively inferior performance to ours, and even sometimes to multi-core CPUs implementations. E.g., for the computation of the Spearman's measure, it took averagely 64 seconds for GPU-based sort-detection algorithm [25] to handle 3mm isotropic resolution (*N*=58523, *L*=512) while <40 seconds were needed for the multi-core CPUs implementation accordingly.

B. Scalability

To assess the scalability of the proposed framework, the elapsed time for constructing large-scale functional networks with varied numbers of networks nodes were demonstrated in Table 1. Sparsity thresholds were specified at 0.1%, 1%, 2%, respectively. In particular, for Kendall's network construction that usually takes longer time than Pearson's and Spearman's, the table only listed the time records within 1000s and corresponding track of N while the framework was certainly applicable to larger scale data sets. The results in Table 1 illustrated that our proposed framework were scalable to high resolution data and could establish voxel-wise functional networks with a large amount of nodes in a short period of time. Specifically, the proposed framework could construct Pearson's or Spearman's networks with 10^6 nodes and 1% edge sparsity using less than 1000 seconds. Considering that only about 200,000 nodes need to be maintained in dense connectomes at an isometric resolution of the 2mm level (Figure 1), our framework is capable of handling high resolution data whose voxel size is far lower than 2mm. Moreover, in actual measurements, it was tested that the framework could handle 1mm voxel-resolution data (*N*=1561152, *L*=1200, 0.1% sparsity threshold) and finish the Pearson's network construction within an hour. To the best of our knowledge, this is the first work enabling the processing of such magnitude data in an acceptable amount of time. Notably, the elapsed time of our network construction algorithms is affected by both node amounts N and sparsity thresholds of networks. The time complexity is $O(N^2)$ approximately, and the distinct network sparsity thresholds mainly affect the time expenditure of CPU-GPU data transfers.

Finally, it was observed that our Pearson's and Spearman's network construction procedures failed when $N = 10^6$ given the 2% sparsity threshold, as a result of inadequate CPU main memory. Actually, the relation between the sparsity threshold and the corresponding quantity of network nodes that our framework could handle is constrained by:

$$N^2 \cdot Sparsity \leq MainMemory/B, \tag{9}$$

where B represented the number of bytes used to store an element (node or edge) in established networks, and $0 < Sparsity \leq 1$. For example, given B = 4 bytes and $Sparsity$ = 2%, our framework supports up to 92×10^4 nodes in the network construction under the current computing environment with 64GB CPU main memory. By contrast, for related works which did not employ the block-wise thresholding strategy, the CPU main memory is required to be large enough to at least hold the entire connectivity matrix, i.e.,

$$N^2 \leq MainMemory/B, \tag{10}$$

in which case the maximum value of N is far less than that in (9). Under the same condition (64GB main memory), at most 16×10^4 nodes are supportable for those works, no matter how sparse the network is. Comparatively, our framework has the better scalability for the increasing number of network nodes.

IV. CONCLUSION AND FUTURE EXPECTATIONS

In summary, this paper provides a scalable GPU-accelerated framework for the fast mapping of dense functional

TABLE I. THE SCALABILITY DEMONSTRATION OF THE PROPOSED FRAMEWORK.

L=128	Sparsity	N=$20 \cdot 10^4$	$40 \cdot 10^4$	$60 \cdot 10^4$	$80 \cdot 10^4$	$100 \cdot 10^4$
Pearson's Measure time:(s)	0.1%	31.90	126.57	278.07	505.76	777.21
	1%	34.47	135.82	307.56	546.13	914.09
	2%	37.36	146.61	324.51	587.20	-
Spearman's Measure time:(s)	0.1%	31.66	124.90	280.03	497.58	782.02
	1%	32.03	133.76	308.60	541.77	917.25
	2%	36.41	145.01	322.00	591.03	-
L=128	Sparsity	N=$10 \cdot 10^4$	$15 \cdot 10^4$	$20 \cdot 10^4$	$25 \cdot 10^4$	$30 \cdot 10^4$
Kendall's Measure time:(s)	0.1%	91.01	210.43	386.50	602.77	861.64
	1%	94.31	217.03	395.61	634.54	914.17
	2%	95.79	223.40	404.97	640.80	951.33

connectomes based on three commonly used FC measures. The proposed framework significantly accelerated the voxel-wise network construction (speedup>50, Kendall's measure) and could scale up to higher voxel-resolution data (<2mm) against related works. We hope that the present study will serve as a building block to facilitate dense connectome studies. In the future, we expect to implement more FC measures under the current framework, especially those measures that enable the detection of the nonlinear, multivariate and frequency-domain connectivity among brain network nodes [30].

ACKNOWLEDGMENT

This work was supported by the Natural Science Foundation of China (Grant Nos. 61373026 and 61622403).

REFERENCES

[1] O. Sporns, G. Tononi, and R. Ktter, "The human connectome: A structural description of the human brain." Plos Computational Biology, vol. 1, no. 4, 2005, p. 42.

[2] D. C. Van Essen and K. Ugurbil, "The future of the human connectome," Neuroimage, vol. 62, no. 2, 2012, pp. 1299–1310.

[3] K. Loewe, S. E. Donohue, M. A. Schoenfeld, R. Kruse, and C. Borgelt, "Memory-efficient analysis of dense functional connectomes," Frontiers in Neuroinformatics, vol. 10, 2016.

[4] Smith et al., "Resting-state fmri in the human connectome project," Neuroimage, vol. 80, 2013, pp. 144–168.

[5] S. Hayasaka and P. J. Laurienti, "Comparison of characteristics between region-and voxel-based network analyses in resting-state fmri data," Neuroimage, vol. 50, no. 2, 2010, pp. 499–508.

[6] V. Essen et al., "The human connectome project: a data acquisition perspective," Neuroimage, vol. 62, no. 4, 2012, pp. 2222–2231.

[7] M. A. de Reus and M. P. Van den Heuvel, "The parcellation-based connectome: limitations and extensions," Neuroimage, vol. 80, 2013, pp. 397–404.

[8] E. Wu and Y. Liu, "Emerging technology about gpgpu," in Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on. IEEE, 2008, pp. 618–622.

[9] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," arXiv preprint arXiv:1202.4347, 2012.

[10] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the gpu–past, present and future," Medical image analysis, vol. 17, no. 8, 2013, pp. 1073–1094.

[11] D. Wu et al., "Making human connectome faster: Gpu acceleration of brain network analysis," in Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on. IEEE, 2010, pp. 593–600.

[12] N. S. Chok, "Pearson's versus spearman's and kendall's correlation coefficients for continuous data," Ph.D. dissertation, University of Pittsburgh, 2010.

[13] S. C. Strother, "Evaluating fmri preprocessing pipelines," IEEE Engineering in Medicine and Biology Magazine, vol. 25, no. 2, 2006, pp. 27–41.

[14] E. Bullmore and O. Sporns, "The economy of brain network organization," Nature Reviews Neuroscience, vol. 13, no. 5, 2012, pp. 336–349.

[15] M. Rubinov and O. Sporns, "Complex network measures of brain connectivity: uses and interpretations," Neuroimage, vol. 52, no. 3, 2010, pp. 1059–1069.

[16] S. L. Simpson, F. D. Bowman, and P. J. Laurienti, "Analyzing complex functional brain networks: fusing statistics and network science to understand the brain," Statistics surveys, vol. 7, 2013, p. 1.

[17] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. ACM, 2009, p. 18.

[18] K. A. Garrison, D. Scheinost, E. S. Finn, X. Shen, and R. T. Constable, "The (in) stability of functional brain network measures across thresholds," Neuroimage, vol. 118, 2015, pp. 651–661.

[19] B. C. Van Wijk, C. J. Stam, and A. Daffertshofer, "Comparing brain networks of different size and connectivity density using graph theory," PloS one, vol. 5, no. 10, 2010, p. e13701.

[20] U. Milic, I. Gelado, N. Puzovic, A. Ramirez, and M. Tomasevic, "Parallelizing general histogram application for cuda architectures," in Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on. IEEE, 2013, pp. 11–18.

[21] C. Croux and C. Dehon, "Influence functions of the spearman and kendall correlation measures," Statistical methods & applications, vol. 19, no. 4, 2010, pp. 497–515.

[22] Y. Wang et al., "A hybrid cpu-gpu accelerated framework for fast mapping of high-resolution human brain connectome," PloS one, vol. 8, no. 5, 2013, p. e62789.

[23] R. Nelsen, "Kendall tau metric," Encyclopaedia of Mathematics, vol. 3, 2001, pp. 226–227.

[24] J. L. Myers, A. Well, and R. F. Lorch, Research design and statistical analysis. Routledge, 2010.

[25] S. Kim, M. Ouyang, and X. Zhang, "Compute spearman correlation coefficient with matlab/cuda," in Signal Processing and Information Technology (ISSPIT), 2012 IEEE International Symposium on. IEEE, 2012, pp. 000 055–000 060.

[26] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," Queue, vol. 6, no. 2, 2008, pp. 40–53.

[27] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," in High-Performance Computing on the Intel® Xeon Phi. Springer, 2014, pp. 167–188.

[28] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," Journal of the American Statistical Association, vol. 61, no. 314, 1966, pp. 436–439.

[29] J. Buckner, J. Wilson, M. Seligman, B. Athey, S. Watson, and F. Meng, "The gputools package enables gpu computing in r," Bioinformatics, vol. 26, no. 1, 2010, pp. 134–135.

[30] F. D. V. Fallani, J. Richiardi, M. Chavez, and S. Achard, "Graph analysis of functional brain networks: practical issues in translational neuroscience," Phil. Trans. R. Soc. B, vol. 369, no. 1653, 2014, p. 20130521.