J Internet Serv Appl (2012) 3:347–357
DOI 10.1007/s13174-012-0071-1

SI: DATA INTENSIVE COMPUTING

Massively parallel non-stationary EEG data processing on GPGPU platforms with Morlet continuous wavelet transform

Ze Deng · Dan Chen · Yangyang Hu · Xiaoming Wu · Weizhou Peng · Xiaoli Li

Received: 27 June 2012 / Accepted: 4 October 2012 / Published online: 10 November 2012

Abstract Morlet continuous wavelet transform (MCWT) has been widely used to process non-stationary electroencephalogram (EEG) data. Nowadays, the MCWT application for processing EEG data is time-sensitive and data-intensive due to quickly increasing problem domain sizes and advancing experimental techniques. In this paper, we proposed a massively parallel MCWT approach based on GPGPU to address this research challenge. The proposed approach treats MCWT as four main computing sub-procedures and parallelizes them with CUDA correspondingly. We focused on optimizing FFT on GPUs to improve the performance of MCWT. Extensive experiments have been carried out on Fermi and Kepler GPUs and a Fermi GPU cluster. The results indicate that (1) the proposed approach (especially on Kepler GPU) can ensure encouraging runtime performance of processing non-stationary EEG data in contrast to CPU-based MCWT, (2) the performance can further be improved on the GPU cluster but performance bottleneck exists when running multiple GPGPUs on one node, and (3) tuning an appropriate FFT radix is important to the performance of our MCWT.

Keywords Morlet continuous wavelet transform · EEG data · GPGPU

Z. Deng · D. Chen (✉) · Y. Hu · X. Wu · W. Peng
School of Computer Science, China University of Geosciences (Wuhan), Wuhan, Hubei, China
e-mail: dan.chen@ieee.org

Z. Deng
e-mail: deng_ze@163.com

Y. Hu
e-mail: allen.huyangyang@gmail.com

X. Wu
e-mail: xiaominwu2012@gmail.com

W. Peng
e-mail: wzpeng.cug@gmail.com

X. Li
National Key Laboratory of Cognitive Neuroscience and Learning, Beijing Normal University, Beijing, China
e-mail: xiaoli.avh@gmail.com

1 Introduction

Most data from either natural phenomena or artificial sources are non-linear and non-stationary in nature [30]. In the past decade, numerous works have focused on how to efficiently analyze non-stationary data [4,14–16,19–21,30,32]. Wavelet transform-based approaches are mainstream for dealing with non-stationary data for their capabilities of multi-resolution analysis for all the scales [17] to extract short-lived transient information.

The approaches in wavelet transform family can be of two types, i.e., continuous wavelet transform (CWT) and discrete wavelet transform (DWT). The most notable difference between CWT and DWT lies in CWT's highly redundant nature of the analyzing functions, which are not limited in an orthonormal basis while DWT selects only those scales which do provide an orthonormal basis. That means CWT's resolution in scale is arbitrary. CWT can then enable a more comprehensive analysis than DWT does, even though CWT must be discretized for numerical evaluation [15]. CWT may rely on various mother wavelets, e.g., Morlet, Mexican hat and Paul [29], among which CWT upon Morlet basis (MCWT) is salient for its capacity of mining transient characteristics hidden in non-stationary data [29] with a wide application in analyzing EEG data [4,16,19–21].

As the problem domain sizes and experimental techniques for recording activities of EEG systems have been advancing quickly, i.e., rapidly increasing number of channels and

sampling frequencies, the density and the spatial scale of non-stationary for EEG data analysis have been increasing exponentially. For instance, for the analysis of the interaction dynamics of multiple neural oscillations [21], the number of electrodes has increased from tens to more than one thousand [4]. For a MCWT application for EEG in practice, it often requires a real-time or near real-time data analysis. As such, nowadays MCWT applications for EEG are commonly time-sensitive and data-intensive.

Modern graphics processing units (GPUs) have evolved from a configurable graphics processor to massively parallel many-core multiprocessors for rapidly solving data and time intensive [24]. In this paper, we explored the feasibility of using General-Purpose computation on Graphics Processing Units (GPGPU) to address the challenge of performance and scalability in EEG MCWT applications. A GPGPU-based MCWT has been developed for this purpose.

We first adapted a MCWT [22] approach to the many-core architecture of GPU. The MCWT algorithm can be viewed as an integration of four computing sub-procedures that are: (1) a forward fast Fourier transform (fFFT) of multi-channel EEG data, (2) transform of Morlet wavelets from time domain to frequency domain, (3) inner production between transformed data and transformed Morlet wavelets, and (4) inversed fast Fourier transform (iFFT) of inner production results.

We employed a coalesced GPU global memory access scheme [12], a parallelization scheme of two-dimension GPU thread grid and one-dimension thread block for the first sub-procedure. Each row of the thread grid owns multiple blocks responsible for one-channel EEG data. For second and third sub-procedures, we dealt with multiple scale factors using a two-dimension GPU thread block assigning method for exploiting time-frequency-scale data parallelism. For the last sub-procedure, we used a similar method to that for the first sub-procedure (fFFT) except that the row number of thread grid needs to be larger since the number of channels of data increases to S times in the case of using S scale factors.

These GPGPU-based algorithms have been designed with NVIDIA CUDA [3] to map these sub-procedures onto four different groups of massively parallel executions. In this study, we further investigated how to enhance the proposed MCWT approach by optimizing FFT using large radix to reduce the time complexity [12].

A case study has been carried out to evaluate the performance of the proposed approach upon GPUs of Fermi [1] architecture and the latest Kepler [2] architecture using a large EEG data. We test this approach on a GPU cluster as well. The results indicate that the proposed approach can ensure encouraging runtime performance of processing non-stationary EEG data in contrast to CPU-based MCWT. To the best of our knowledge, the proposed approach is the first massively parallel CWT aided by GPGPU. It should also be noted that the approach applies to other types of non-stationary data rather than being specific to EEG.

The remainder of this paper is organized as follows: Sect. 2 presents some typical work related to processing non-stationary data. Section 3 introduces the conventional MCWT on CPU and proposes our GPGPU-based MCWT and its variants. Experiments and results are given in Sect. 4. We conclude the paper with a summary and present future work in Sect. 5.

2 Related work

In the past decade, numerous methods have emerged for processing non-stationary data [4,8,9,13–16,19–21,23,25–27,30,32]. Short-Time Fourier Trans-form (STFT) and Wigner–Ville distribution (WVD) [14] are two classic methods. STFT uses uniform time and frequency resolutions to analyze non-stationary signals while WVD can extract cross-terms between various components of non-stationary signals. A common problem with the two methods is that they often fail to discover transient phenomena from non-stationary data.

In addition to the above, Wang et al proposed an adaptive analysis method called empirical data decomposition (EDD) [30]. The EDD algorithm also implements a multi-resolution analysis similar to wavelet transform. Xuan et al. [32] used empirical mode decomposition to decompose precipitation data to probe the non-stationary dynamics.

Comparing to these approaches, the wavelet transform-based methods have a more wide range of applications. For example, Gurley and Kareem proposed a series of methods based on CWT and DWT to process non-stationary data in terms of earthquake, wind and ocean engineering [13]. Akhtar et al. developed a framework based on independent component analysis (ICA) and a DWT variant to correctly detect artifacts embedded in EEG data [4]. Using the proposed framework, ICA is used to extract artifact-only independent components from EEGs and further the DWT is employed to remove any cerebral activity from the extracted artifacts independent components to get clean EEG data. The key difference from our work is that our work focuses on CWT with more comprehensive ability of analyzing non-stationary data.

MCWT has been used in various research areas and disciplines. Li et al. [20,21] applied MCWT in analyzing and quantifying the instantaneous interaction dynamics between neuronal population to understand the mechanism of epileptic seizure in EEG. Klein et al. [16] proposed a MCWT-based coherence analysis approach to monitoring time-dependent changes in the coherences among multi-channel EEG. Fligge et al. [9] used MCWT to objectively determine the length of sunspot cycle and carry out error analysis on long-term solar

activities, e.g., sunspot number, sunspot area. MCWT has also been employed to extract two complementary wavelet skeleton spectra to discriminate the components of periodicities and of hierarchies of discontinuities from several large-size time series represent solar activity records [27]. Pi$\ddot{o}$ft [26] has used MCWT to process four long Czech mean monthly temperature series from 1775 to 2001, and then the temperature variability in the Czech Republic has been examined. None of them have considered to use GPU platforms to help MCWT while we did so.

Recent research has focused on using many-core platform such as GPU to improve wavelet transform, and nearly all works along this direction are targeted at DWT. For instance, Wong et al implement a two-dimension DWT with Cg and OpenGL on a GeForce GTX 7800 [31]. Similarly, in [28] authors also explore the implementation of a fast 2D-DWT with Filter Bank Scheme (FBS) and Lifting Scheme (LS) using Cg on the same GPU. With NVIDIA's CUDA library [6], people have implemented 2D-DWT variants [10,18] and a 3D-DWT on GPUs [11]. In contrast to these methods, we aimed to significantly promote a CWT approach with the latest GPGPU technologies to better cater for the needs of processing massive non-stationary EEG data.

3 Morlet continuous wavelet transform on GPGPU

In this section, we first present a MCWT algorithm operation on CPU. We then detail the design of the GPGPU-based MCWT for multi-channel EEG data.

3.1 Morlet continuous wavelet transform on CPU

Let us denote non-stationary EEG data be a discrete time series $X = \{x_n | n \in [0, N]\}$ with equal time space dt. A CWT algorithm computes wavelet coefficients $\omega(s, \tau)$ of the discrete time series X. Based on [21], $\omega(s, \tau)$ is computed by the convolution of wavelet function $\psi(n)$ analyzed series X, $\langle X, \psi_{\tau,s}(n)\rangle$, that means:

$$w(s,\tau) = \langle X, \psi_{\tau,s}(n)\rangle = \sum_{n=0}^{N-1} x_n \psi^* \left(\frac{(n-\tau)dt}{s}\right) \quad (1)$$

where s and τ represent the scale and translation, respectively and "*" means complex conjugation. By tuning the value of scale factor s, we can extract a set of various frequency components. Through a CWT, the information of X can then is projected into a two-dimension space (s and τ) for further data analysis. As for MCWT, the parent wavelet function $\psi_0(n)$ in CWT is given:

$$\psi_0(n) = \pi^{-\frac{1}{4}} e^{j\omega_0 n} e^{-\frac{1}{2}n^2} \quad (2)$$

where ω_0 is the wavelet central angle frequency. Furthermore, based on the time-domain convolution theorem [5], the convolution of two series in time domain can be indirectly computed with inner production of the transformed frequency series of two series. The convolution $\langle X, \psi_{\tau,s}(n)\rangle$ can be computed as the following four steps:

- *Step1*. Transform X from time domain to frequency domain to generate a frequency series $X(\omega)$ using Fourier transform.
- *Step2*. Transform $\psi_{\tau,s}(n)$ from time domain to frequency domain to generate a frequency series $\phi^*(s\omega)$ with computing angle frequency.
- *Step3*. Compute the inner production of $X(\omega)$ and $\sqrt{s}\ \phi^*(s\omega)$, that is referenced as $[X(\omega)\ ,\ \sqrt{s}\ \phi^*(s\omega)]$, where $\sqrt{s}$ is a factor for energy normalization across the different scales.
- *Step4*. Transform $[X(\omega)\ ,\ \sqrt{s}\ \phi^*(s\omega)]$ from frequency domain back to time domain to get $w(s, \tau)$ using inverse fourier transform.

Thus, $w(s, \tau)$ can be computed as the following formula:

$$w(s,\ \tau) = \mathrm{IFT}([X(\omega), \sqrt{s}\phi^*(s\omega)]) \quad (3)$$

where IFT means the inverse fourier transform. More specifically, the algorithm of MCWT on CPU (namely "MCWT-CPU") for multiple-channel EEG data is shown in Algorithm 1.

Taking an EEG data processing as instance: an EEG data set has m channels and n (assume a number of power of 2) data points per channel. The EEG data can be written as a matrix $E[m][n]$, and we denote l scale factors as a vector $S[l]$. Since the data segment in each channel can be scaled up to l times after being processing with MCWT, the wavelet coefficients are written as a three-dimension array $W[l][m][n]$ as illustrated in Fig. 1.

3.2 GPU-based Morlet continuous wavelet transform (MCWT-GPU)

Based on Algorithm 1, parallelism exists in the following main sub-procedures:

- *1*. Forward 1D-FFT (fFFT) procedure of multiple-channel EEG data (lines 2–4 in Algorithm 1).
- *2*. The transform procedure of Morlet wavelets from time domain to frequency domain under multiple channels and different scale factors (lines 5–11).
- *3*. Inner production of $X(\omega)$ and $\sqrt{s}\ \phi^*(s\omega)$ (lines 12–18).
- *4*. Inverse FFT (iFFT) procedure of multiple-channel data under different scale factors (lines 19–23).

We proposed a parallelized MCWT (namely "MCWT-GPU") as in Algorithm 2.

Algorithm 1: MCWT-CPU

Procedure MCWT_CPU(*E*[*m*][*n*], *S*[*l*], ω_0, *dt*, *W*[*l*][*m*][*n*]) /* Input : E is a matrix stores m-channel EEG data, S is a vector with a set of scale factors, ω_0 means wavelet central angle frequency, dt is EEG signal sampling period time */
/* Output: W is MCWT's results */
/*Transform each channel's EEG data from time domain to frequency domain by forward fast Fourier transform */
for i =*1;*i < m;i ++ **do**
  FFT_CPU($E[i][]$,*forward*)
end
/*Transform Morlet wavelets $\psi_{\tau,s}$ to $\phi^*(s\omega)$ and store to $W[l][m][n]$ */
for i = *1;* i < l;i ++ **do**
  /* l is the number of scale factors */
  for j = *1;* j < m;j ++ **do**
    /* m is the number of channels */
    for k = *1;* k < n;k ++ **do**
      /* n is length of data per channel */
      $W[i][j][k]$ = compute_ ϕ^* ($S[i]$, k, ω_0 , dt, n)
    end
  end
end
/*Compute the inner production of $X(\omega)$ and $\sqrt{s}\phi^*(s\omega)$ */
for i = *1;* i < l;i ++ **do**
  for j = *1;* j < m;j ++ **do**
    for k = *1;* k < n;k ++ **do**
      $W[i][j][k]=E[j][k]\times\sqrt{s[i]}\times W[i][j][k]$
    end
  end
end
/* Transform each channel's EEG data from time domain to frequency with inverse fast Fourier transform */
for i = *1;* i < l;i ++ **do**
  for j = *1;* j < m;j ++ **do**
    FFT_CPU($W[i][j][]$, *inverse*)
  end
end
Procedure compute_ ϕ^*(*s*, *i*, ω_0 , *dt*, *n*) /*Input: s is a scale factor i is index of some element in one-channel EEG array, n is the data length of one-channel EEG array */
/* Output: transformed frequency value */
Calculate an angle frequency with $f = \frac{i}{n} \times \frac{2\pi}{dt}$
Calculate frequency value of Morlet wavelet with $\phi = \pi^{-1/4} \times \sqrt{\frac{2\pi}{n \times dt}} \times e^{\frac{-(sf-\omega_0)^2}{2f}}$
return ϕ

3.2.1 Parallelizing sub-procedures based on FFT

One of the key issues of a GPU-based algorithm is the design of thread blocks to maximize the exploitation of data parallelism. Sub-procedure (1) deals with multi-channel data using fFFT while sub-procedure (4) executes iFFT to handle multi-channel data under various scale factors. Two schemes of thread assignment have been proposed for parallelizing fFFT and iFFT as illustrated in Fig. 2.

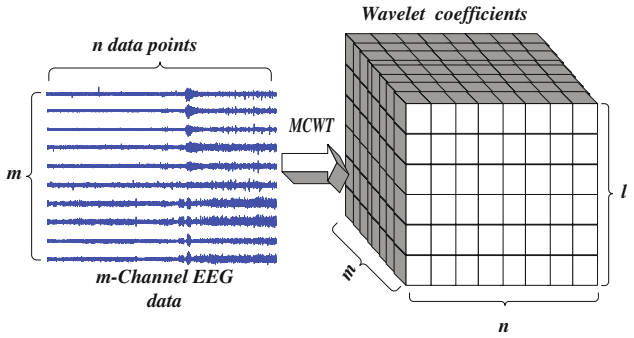


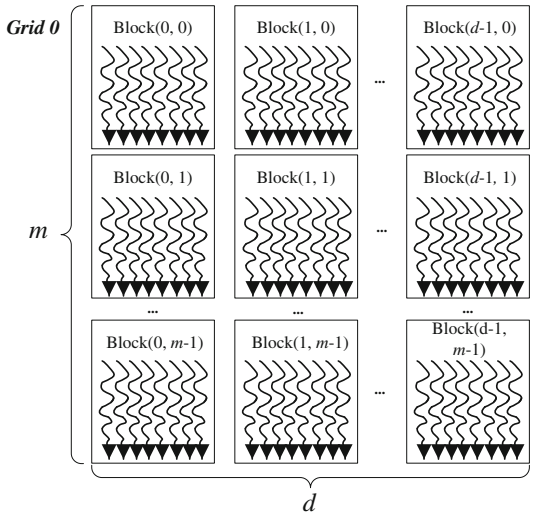
Fig. 1 Computing wavelet coefficients by MCWT

The first scheme (see Fig. 2a) makes all thread blocks two-dimensional. In each block, all threads are indexed along one dimension. Section 4.1 details how the number of thread in one block may affect the performance. For fFFT, a thread block $b(x, y)$ is responsible for processing the xth segment's data of the yth channel with $x \in [0, d)$ and $y \in [0, m)$. For iFFT, all thread blocks are also two-dimensional, but a thread block should be referenced as $b(x, y\times z)$ to deal with the xth segment's data of the yth channel under a scale factor $z \in [0, s)$ (see 2(b)).

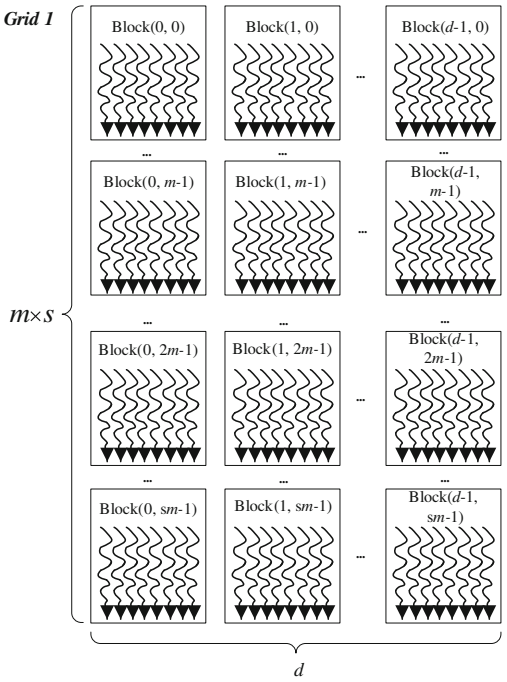
NVIDIA has provided CUFFT, which is an official parallel FFT library on GPU [7]. An alternate FFT has also been proposed in [12], which optimizes FFT's performance via three efficient memory access schemes:(1) shared memory is used when the size of data set is small, (2) coalesced global memory access scheme is employed for large size, and (3) a hierarchical memory access method is to compute large data stream's FFT by combining the FFTs of smaller data with share memory. In this paper, we use the coalesced global memory access scheme to develop the GPGPU-aided MCWT.

Algorithm 2: MCWT-GPU

Procedure MCWT_GPU($E[m][n]$, $S[l]$, ω_0, dt, $W[l][m][n]$) /* Input : E is a matrix stores m-channel EEG data, S is a vector with a set of scale factors, ω_0 means wavelet central angle frequency, dt is EEG signal sampling period time */
/* Output: W is MCWT's results */
/*Transform each channel's EEG data from time domain to frequency domain by forward fast Fourier transform */
foreach *channel's EEG data* ***in parallel*** **do**
  FFT_GPU($E[i][]$, *forward*) *where* $i \in [0, m)$
end
/*Transform Morlet wavelets $\psi_{\tau,s}$ to $\phi^*(s\omega)$ and store to $W[l][m][n]$ */
foreach *element in the three-dimension array* W ***in parallel*** **do**
  $W[i][j][k]$ = compute_$\phi^*(S[i], k, \omega_0, dt, n)$ *where* $i \in [0, l) \wedge j \in [0, m) \wedge k \in [0, n)$
end
/*Compute the inner production of $X(\omega)$ and $\sqrt{s}\phi^*(s\omega)$ */
foreach *element in the three-dimension array* W ***in parallel*** **do**
  $W[i][j][k]=E[j][k]\times\sqrt{s[i]} \times W[i][j][k]$ /*inner product */
end
/* Transform each channel's EEG data from time domain to frequency with inverse fast Fourier transform */
foreach *channel's data with one scale factor* ***in parallel*** **do**
  FFT_GPU($W[i][j][]$, *inverse*) *where* $i \in [0, l)$ and $j \in [0, m)$
end
Procedure compute_ ϕ^*(s, i, ω_0 , dt, n) /*Input: s is a scale factor i is index of some element in one-channel EEG array, n is the data length of one-channel EEG array */
/* Output: transformed frequency value */
Calculate an angle frequency with $f = \frac{i}{n} \times \frac{2\pi}{dt}$
Calculate frequency value of Morlet wavelet with $\phi = \pi^{-1/4} \times \sqrt{\frac{2\pi}{n\times dt}} \times e^{\frac{-(sf-\omega_0)^2}{2f}}$
return ϕ



(a) The scheme of thread for fFFT



(b) The scheme of thread assignment for iFFT

Fig. 2 Thread assignment schemes for sub-procedures 1 and 4 (fFFT and iFFT)

3.2.2 Parallelizing other sub-procedures

The other sub-procedures, i.e., transforming Morlet wavelets from time domain to frequency domain and inner production, are similar to each other. We then designed one thread assignment scheme for both sub-procedures as shown in Fig. 3.

Similar to the design of thread blocks for fFFT, here the scheme first sets up $d\times$ m two-dimensional thread blocks in a grid. A total number of d blocks cooperate with each other to process the data in one channel. In the context of a thread block, the scheme indexes intra-block threads along two dimensions to deal with data with different scale factors. Taking *Thread*(x, y) as an example, the first dimension x denotes the xth data segment while the second dimension y means that the data is processed under the scale factor y.

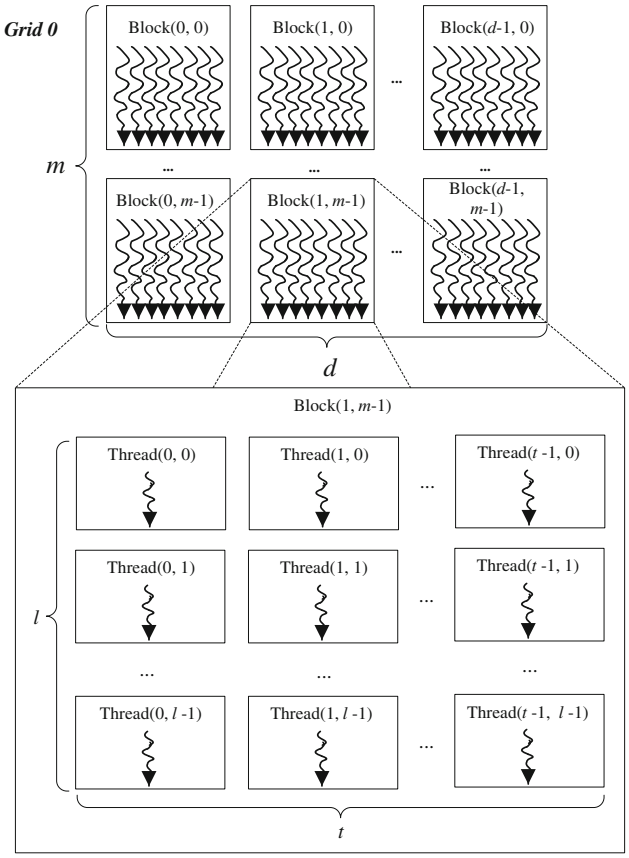


Fig. 3 The thread assignment scheme for Morlet wavelet frequency transform and inner production

3.2.3 Parallelism analysis

Given a fixed-size workload with m-channel EEG data, we analyze the speedup ratio of proposed MCWT. Let the time cost of processing the fixed-size workload using CPU and GPU be T_{CPU} and T_{GPU} respectively. Thus, the speedup factor is that:

$$\mathrm{SpeedUp} = \frac{T_{\mathrm{CPU}}}{T_{\mathrm{GPU}}}$$

Based on Algorithm 1. T_{CPU} is the total time of processing m-channel EEG data in sequence. So:

$$T_{\mathrm{CPU}} = \sum_{i=1}^{m} (\mathrm{FFT} + \mathrm{Transfer} + \mathrm{InnerProduct} + \mathrm{iFFT})$$

where *FFT*, *Transfer*, *InnerProduct* and *iFFT* represent the time to execute four sub-procedures for one-channel data.

Algorithm 3: Global memory FFT on GPU

```
1  Procedure Call_FFT_GPU(N, R< 2 >, *dataI, *dataO)  /*Input: N is the
     number of EEG points in a channel, R is FFT's radix < 2 >, dataI is the pointer of
     input array                                                          */
   /* Output: dataO is the point of output array                         */
2  Initialize thread grid and block variables(namely dimgrid and dimblock) via thread
   assignment schemes in figure 2
3  for Ns = 1; Ns < N; Ns*=R do
4  |   FFT_GPU<<< dimgrid, dimblock >>> (N, R, Ns, dataI, dataO)
5  |   cudaThreadSynchronize()
6  |   Swap dataI and dataO
7  end
8  return dataO
9  Procedure FFT_GPU(N, R< 2 >, Ns, input, output )  /*Ns: data length for each
     FFT iteration                                                        */
   /*compute the thread index                                           */
10 inx = blockIdx.x× blockDim.x + threadIdx.x
11 FFTIteration(inx, N, R, Ns, input, output)
   /*FFT iteration running                                              */
12 Procedure FFTIteration(inx, N, R< 2 >, Ns, input, output)
13 float2 v[R]
14 float angle = -2× PI ×(inx%Ns)/(Ns × R)
15 for r = 0, r < R, r++ do
16 |   v[r] = input[inx + r × N/R]
17 |   v[r]× = (cos(r × angle), sin(r×angle))
18 end
   /*FFT core computing procedure on GPU                                */
19  FFT< R = 2 >(v)
20 Store v into output array and return output
```

Let N_c be the number of cores in one GPU card. T_{GPU} consists of two parts. One part is the parallel execution time using GPU for processing some data channel with the maximum time cost. The other is time for GPU thread synchronization. So, T_{GPU} can be represented as:

$$T_{\text{GPU}} = \frac{\max_{m \text{ channels}}(\text{FFT} + \text{Transfer} + \text{innerProduct} + \text{iFFT})}{\alpha} + \text{Sync}$$

where α means the number of available GPU cores and $\alpha \in [1, N_c]$. The value of α depends on how many GPU cores can be exploited to execute MCWT in parallel. *Sync* is time spent to synchronize the GPU threads.

3.2.4 Optimizing FFT on GPU

We further investigated how to improve the GPGPU-aided approach by optimizing the FFT algorithm. Algorithm 3 illustrates the global memory FFT for processing one-channel EEG data as suggested in [12]. The value of the FFT's radix is set to 2.

In Algorithm 3, FFT_GPU() function needs to be called $\log_R^N$ times for processing the data with the length of *Ns* (line 3).

Since each call will lead to high data transferring costs, using bigger FFT radix *R* can reduce the times of calling Call_FFT_GPU(). It is a way to improve FFT especially for large-size data. However, the algorithm only described the implementation of radix-2 FFT (line 18). Therefore, this motivates us to propose a FFT with radix-*R* (*R*>2) on GPU. This algorithm is described in Algorithm 4 where float2 is used to represent a complex number.

Through the above FFT algorithm's modifications, we further accelerate MCWT on GPGPU for larger-size data.

4 Experiments and results

We have evaluated the performance of the proposed MCWT methods against large EEG dataset upon various platforms empowered by cutting-edge NVIDIA GPUs and

Algorithm 4: radix-R FFT on GPU

```
1  Procedure FFT(float2 *v) template< int R > float2 v0[R], v1[R]
2  Copy v[R] to v0[R]
   /* assign a thread                                              */
3  int idx = blockIdx.y
4  while idx < R do
5      v[idx] = 0
6      foreach element k in array v in parallel do
7          v1[k].x = _cosf(2 × k × idx × PI/R)
8          v1[k].y = -1 × _sinf(2 × k × idx × PI/R)
9          v[idx].x += multi(v0[k], v1[k]).x
10         v[idx].y += multi(v0[k], v1[k]).y
11     end
12     idx += blockDim.y
13 end
```

high-performance networks. The experiments focus on execution times of these GPGPU-aided MCWT methods.

4.1 Experiment setup

4.1.1 Data set

We chose an EEG dataset for testing, which was obtained from a patient with epilepsy using 64 sampling channels with a frequency of 1,600 Hz for one hour. In total, the EEG has $64 \times 3{,}600 \times 1{,}600$ data points.

Given that a MCWT uses M scale factors, each point in the EEG signal needs to occupy at least $2\times$ M times GPU memory space. This is because the value of a data point first should be transformed to a complex number and then be scaled M times when being processed over a GPU. As a result, a GPGPU-aided MCWT algorithm is likely to cause the GPU memory to deplete.

4.1.2 Testbed configurations

We chose two Fermi GPUs and a Kepler GPU to evaluate the GPGPU-aided methods on top of three individual computers. The Table 1 gives the major configurations of the three computers.

The proposed approach has also be evaluated on a GPU cluster illustrated in Fig. 4. The GPU cluster consists of one master node for managements and three slave nodes for executing computing tasks. These four nodes are connected via one 1Gbps's Ethernet and a 40Gbps's InfiniBand network and each node is equipped with four Tesla C2050 GPU cards, eight E5620 CPU cores and 24GB host memory.

4.1.3 Specification of the number of threads in a block

We have first explored whether/how the number of threads in a thread block may affect the runtime performance for a given scale on a GPU. For this purpose, we executed MCWT-GPU to process a data segment over a GPU and tuned the number of threads to measure the execution times. As suggested in [3], the numbers of threads per thread block have been specified as powers of two and 2^{10} to the maximum. Figure 5 highlights the results with scale set as 20 on GTX580, Tesla C2025, and GTX680. For the three GPUs, the numbers of threads to achieve the optimal performance are 2^7, 2^8, and 2^9 respectively. In the experiments for examining runtime

Table 1 Major hardware and software features of experimental computers

Features	Computer#1	Computer#2	Computer#3
Hardware			
CPU	E7500@2.93GHz	E5620@2.4GHz	i7-2600@3.4GHz
GPU	GTX580 (Fermi)	Tesla C2050 (Fermi)	GTX680 (Kepler)
Memory of the Host	6GB	24GB	16GB
Software			
OS	Windows 7	Red Hat Enterprise Linux Server 5.4	Windows 7
CUDA version	4.1	4.1	4.1

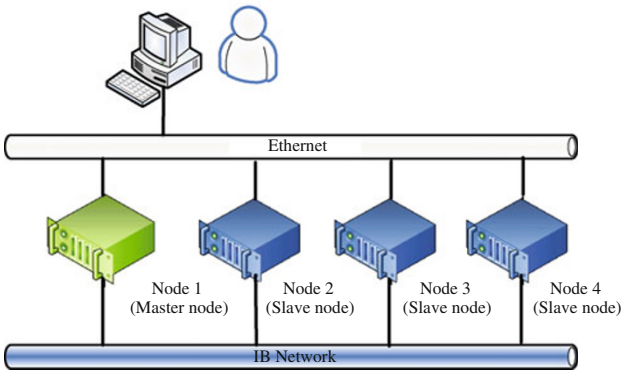


Fig. 4 The Fermi GPU cluster

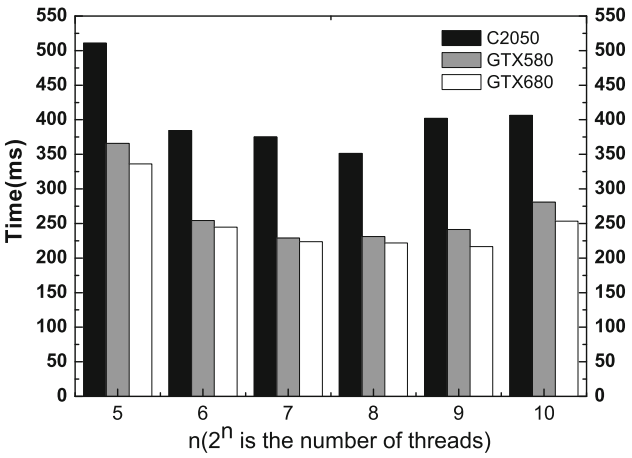


Fig. 5 Experimental results for determining the number of threads in a thread block over various GPUs with scale = 20

Table 2 Performance comparison of MCWT-C and MCWT-SG in terms of executing time

Scale	MCWT-C(s)	MCWT-SG-C(s)	Speedup	MCWT-SG-O(s)	Speedup
Computer#1 (GTX580)					
10	4,513	486	9.3	47	96.0
20	8,719	702	12.4	77	113.2
30	12,938	1,024	12.6	108	119.8
40	17,160	1,568	11.0	136	126.2
50	21,244	1,924	11.0	167	127.2
60	25,067	2,419	10.4	198	126.6
70	29,039	2,871	10.1	230	126.3
Computer#2 (Tesla C2050)					
10	2,342	292	8.0	71	32.9
20	4,611	570	8.1	144	32.1
30	6,937	939	7.4	202	34.3
40	9,357	1,344	7.0	267	35.1
50	11,654	1,795	6.5	330	35.3
60	13,991	2,216	6.3	388	36.1
70	16,328	2,702	6.1	455	35.9
Computer#3 (GTX680)					
10	2,987	228	13.1	39	76.6
20	5,876	516	11.4	67	87.7
30	8,753	951	9.2	94	93.1
40	11,619	1,345	8.6	121	96.1
50	14,375	1,774	8.1	149	96.5
60	17,109	2,253	7.6	178	96.1
70	20,508	2,719	7.5	206	99.6

efficiency, we have always identified the number of threads for each scale over a given GPU and used the optimal setting for performance evaluation.

4.2 Runtime efficiency

In the following experiments, we first compared the execution times for processing the whole EEG data with MCWT-GPU based on CUFFT and optimized FFT in [12] on a single GPU called as MCWT-SG-C and MCWT-SG-O respectively against the sequential MCWT method based on CPU(MCWT-C) as introduced in Sect. 3.1. We also measured the runtime times for processing this EEG data set upon the GPU cluster (MCWT-GC). Finally, we evaluated the performance of optimized GPGPU-aided MCWT.

4.2.1 Executing times on single GPUs

For this set of experiments, the executing time includes the time required to copy data pieces from host to device and vice versa. The parameter ω_0 of MCWT is set to 6 as an optimal value to adjust the time-frequency resolution [21].

As shown in Table 2, Comparing to MCWT-C, MCWT-SG-C gains speedups ranging from 9.3 to 10.1 on computer #1 , from 8.0 to 6.1 on computer #2 and from 13.1 to 7.5 on computer #3 with the increase of scale. This can be seen that MCWT-SG-C faces a performance bottleneck when processing large-size data.

In contrast, MCWT-SG-O gains higher speedups than MCWT-SG-C. When scale = 40, MCWT-SG-O runs 11.5, 5.3 and 11.1 times of MCWT-SG-C. Clearly, MCWT-SG-O dramatically outperforms MCWT-SG-C for dealing with relatively large data. We trust that the performance improvement is a result of using optimized FFT. In other words, FFT is a key factor which contributes to the overall performance of MCWT. With the assistance of the coalesced global memory access scheme, MCWT-SG-O needs to access the global memory much less than its counterpart does especially when dealing with large data.

MCWT-C performs better on Computer #2 than on Computer #3. This is because Computer #2 has a much larger

main memory to be capable of handling the high demand for memory by the MCWT algorithm.

MCWT-SG-O performs better on Computer #3 than on other platforms. This is because GTX 680 adopts Kepler, the latest GPU architecture. The memory subsystem of the Kepler architecture is completely revamped, which results in a 6008MHz data rate. GTX 680 offers the highest memory clock speeds of any GPU in the industry [2]. In this way, the proposed method using optimized memory access scheme can properly exploit the new features of memory subsystem of the Kepler GPU. The results again indicate that it is important to in order to improve the performance of GPGPU-aided MCWT.

4.2.2 Executing times on the GPU cluster

In this set of experiments, we equally distributed the whole EEG data set into three slave nodes of the GPU cluster and processed them in parallel. On each slave node, the data are simultaneously processed using GPU-aided MCWT with various numbers of GPUs. In our setting, master node means management node to control parallel tasks while slave nodes are computing nodes.

For comparison purpose, the results of MCWT-SG on computer #1 with a Tesla C2050 GPU Card (shown in Table 1) have been referred to as a baseline.

In Fig. 6, MCWT-GC-2(3, 4)GPUs/Node means each node uses two(three, four) GPUs to run MCWT-GC-O. The results indicate that MCWT-GC-2GPUs/Node has a speed-up of 2.6 comparing to MCWT-SG-O. This is reasonable as more nodes were used to process the data set in parallel.

However, we observed that MCWT-GC-3GPUs/Node only has 1.4 times than MCWT-GC-2GPUs/Node and MCWT-GC-4GPUs/Node is 1.1 times relative to MCWT-GC-3GPUs/Node. That is because that multiple GPUs on the same node contend PCI-E Bus resources.

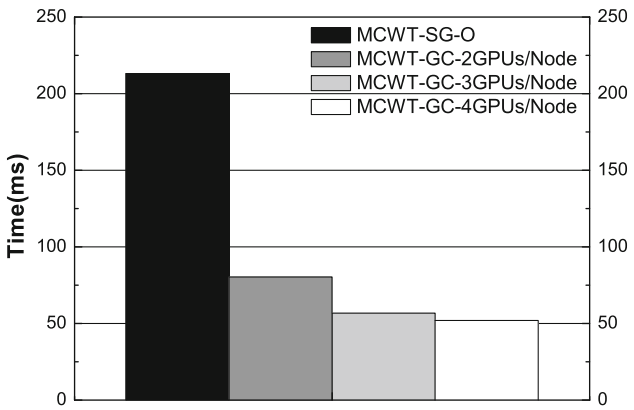


Fig. 6 The executing time comparison between MCWT-SG-O and MCWT-GC with scale = 40

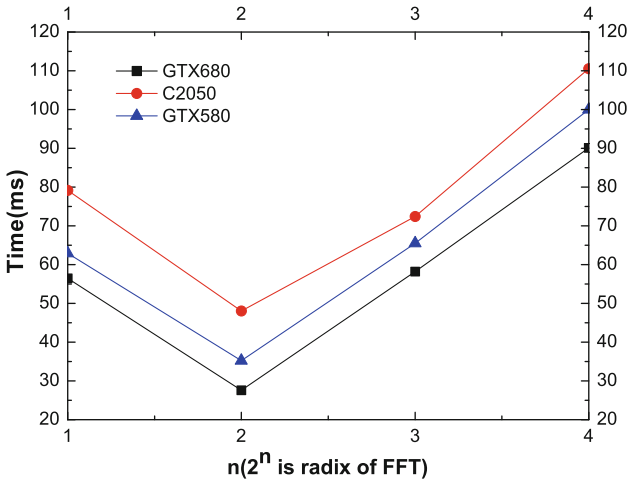


Fig. 7 The executing times of MCWT-SG-O with various FFT radixes under scale = 30

4.2.3 Evaluating the improved GPGPU-aid MCWT

To study the effect of the FFT with a larger radix on MCWT, we executed MCWT-SG-O multiple times to process an EEG data segment (6 s) under scale 30, and each run has a different radix setting for the FFT.

Figure 7 shows that MCWT-SG-O performs better when using radix-4 FFT than using radix-2 FFT. However, MCWT-SG-O's performance becomes worse when FFT radix increases from 4 to 8 and 16. The results indicate that selecting an appropriate radix value is important to MCWT-SG-O's performance. When a large radix value is set, although MCWT iterates few times, the workload of a single iteration may become excessively heavy. If the workload exceeds the parallel processing capacity of the GPU, the performance of MCWT-SG-O will certainly decrease.

5 Conclusion and future work

In this paper, we proposed a parallel MCWT with GPGPU platforms to address the challenge of analyzing massive non-stationary EEG data in an efficient and scalable manner.

The proposed approach adapts the embedded parallelisms in the MCWT algorithm to the many-core architecture of GPU using CUDA platform. The MCWT algorithm has been separated into four main sub-procedures. The sub-procedures have been parallelized using various schemes, including FFT, transforming Morlet wavelets from time domain to frequency domain, and inner production. The improved version of GPGPU-aided approach has been proposed as well.

A case study of EEG data analysis has also been performed to examine the proposed approach and its performance. Different GPUs have been adopted for the purpose, including

devices of Fermi and Kepler architectures. Furthermore, we also evaluated our approach on a GPU cluster. Finally, we assessed the impacts of FFT radix on GPGPU-aided MCWT.

Experimental results show that (1) MCWT-SG-O can significantly outperform MCWT-C and MCWT-SG-C, especially on Kepler GPU, (2) MCWT-GC can further improve the performance but performance bottleneck exists when running multiple GPGPUs on one node, and (3) tuning an appropriate FFT radix is important to MCWT-SG-O.

In the future, we plan to further study the approach to solving bottleneck of MCWT-GC for multiple GPGPUs on one node.

Acknowledgments This work is funded in part by National Science Fund for Distinguished Young Scholars (grant No.61025019), the National Natural Science Foundation of China (grants No. 61272314), the Program for New Century Excellent Talents in University (NCET-11-0722), the Fundamental Research Funds for the Central Universities (No.CUGL100608, No.CUGL100231, No.G1323511175 and G1323521289, China University of Geosciences, Wuhan), the Specialized Research Fund for the Doctoral Program of Higher Education (grant No. 20110145110010), the Programme of High-Resolution Earth Observing System (China), and the Hundred University Talent of Creative Research Excellence Programme (Hebei, China).

References

1. NVIDIA Corporation. Nvidia next generation cuda compute architecture: Fermi (2009)
2. NVIDIA Corporation. Whitepaper NVIDIA GeForce GTX 680 (2012)
3. NVIDIA CUDA C programming guide version 4.2 (2012)
4. Akhtar MT, Mitsuhashi W, James CJ (2012) Employing spatially constrained ICA and wavelet denoising, for automatic removal of artifacts from multichannel EEG data. Signal Process 92(2):401–416
5. Boashash B (2003) Time frequency signal analysis and processing: a comprehensive reference
6. Corporation N (2007) Nvidia compute unified device architecture (CUDA) programming guide version 1.1
7. Corporation N (2012) CUDA CUFFT, Library
8. Erol S (2011) Time-frequency analyses of tide-gauge sensor data. Sensors 11:3939–3961
9. Fligge M, Solanki SK, Beer J (1999) Determination of solar cycle length variations using the continuous wavelet transform. Astron Astrophys 483:313–321
10. Franco J, Bernabe G, Fernandez J, Ujaldon M (2011) The 2D wavelet transform on emerging architectures: GPUs and multicores. J Real-Time Image Process 99:1–8
11. Francoa J, Bernab G, Fernandez J, Ujaldon M (2010) Parallel 3D fast wavelet transform on manycore GPUs and multicore CPUs. In: International conference on computational science, p 1101C1110
12. Govindaraju NK, Lloyd B, Dotsenko Y, Smith B, Manferdelli J (2008) High performance discrete Fourier transforms on graphics processors. In: Proceedings of supercomputing, pp 1–12
13. Gurley K, Kareem A (1999) Applications of wavelet transforms in earthquake. Wind and ocean engineering. Eng Struct 21(2):149–167
14. Hambaba A (2012) Nonstationary statistical tests in time-scale space. In: IEEE aerospace conference proceedings, pp 373–379
15. Johnson RW (2012) Symmetrization and enhancement of the continuous Morlet transform for spectral density estimation. Int J Wavelets Multiresol Inf Process 10(1):1–12
16. Klein A, Sauer T, Jedynak A, Skrandies W (2006) Conventional and wavelet coherence applied to sensory-evoked electrical brain activity. IEEE Trans Biomed Eng 53:266–272
17. Kumar P, Foufoula-Georgiou E (1997) Wavelet analysis for geophysical applications. Rev Geophys 35(4):385–412
18. van der Laan WJ, Jalba AC, Roerdink JB (2011) Accelerating wavelet lifting on graphics hardware using CUDA. IEEE Trans Parallel Distrib Syst 22(1):132–146
19. Lachaux JP, Lutz A, Rudrauf D, Cosmelli D, Quyen MLV, Martinerie J, Varela F (2002) Estimating the time-course of coherence between single-trial brain signals: an introduction to wavelet coherence. Clin Neurophysiol 32:157–174
20. Li X, Yao X, FIEEE JRGJ, Fox J (2005) Computational neuronal oscillation with morlet wavelet transform. In: Proceedings of 27th annual international conference of the IEEE engineering in medicine and biology Society, pp 1–4
21. Li X, Yao X, Fox J, Jefferys JG (2007) Interaction dynamics of neuronal oscillations analysed using wavelet transforms. J Neurosci Methods 160:178–185
22. Liu CL (2010) A tutorial of the wavelet transform
23. SouzaEcher MP, Echer E, Nordemann DJR, Rigozo NR (2009) Multi-resolution analysis of global surfaceair temperature and solar activity relationship. J Atmos Solar-Terrestr Phys 71:41–44
24. Nickolls J, Dally WJ (2010) The GPU computing era. IEEE Micro 30(2):56–69
25. Park SG, Sim HJ, Lee HJ, Oh JE (2008) Application of non-stationary signal characteristics using wavelet packet transformation. J Mech Sci Technol 22(11):2122–2133
26. Pioft P, Kalvova J, Brazdil R (2004) Cycles and trends in the CZECH temperature series using wavelet transforms. Int J Climatol 24:1661–1670
27. Polygiannakis J, Preka-Papadema P, Moussas X (2003) On signal-noise decomposition of timeseries using the continuous wavelet transform: application to sunspot index. Astr Soc 343:725–734
28. Tenllado C, Setoain J, Pinuel L, Tirado F (2008) Parallel implementation of the 2D discrete wavelet transform on graphics processing units: Filter Bank versus Lifting. IEEE Trans Parallel Distrib Syst 19(3):299–310
29. Torrence C, Compo GP (1998) A practical guide to wavelet analysis. Bull Am Meteorol Soc 79(1):61–78
30. Wang X, Deng J, Wang Z, Li C (2007) An adaptive analysis method for non-stationary Data_Empirical data decomposition. In: Third international conference on natural computation, pp 3–7
31. Wong TT, Leung CS, Heng PA, Wang J (2007) Discrete wavelet transform on consumer-level graphics hardware. IEEE Trans Multimedia 9(3):668–673
32. Xuan Z, Xie S, Sun Q (2010) The empirical mode decomposition process of non-stationary signals. In: International conference on measuring technology and mechatronics automation, pp 866–869