2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

A High Performance Computing Method for Noise Cross-Correlation Functions of Seismic Data

Junwei Zhou*, Qian Wei†, Chao Wu‡¶, Guangzhong Sun§
*School of Earth and Space Sciences, University of Science and Technology of China
†School of Physical Sciences, University of Science and Technology of China
‡Network and Information Center, University of Science and Technology of China
§School of Computer Science and Technology, University of Science and Technology of China
Email:{zjw330501, wq0320}@mail.ustc.edu.cn, {wuchao15, gzsun}@ustc.edu.cn
¶Corresponding Author

***Abstract*—Calculation of noise cross-correlation functions (NCF) plays an important role in ambient noise imaging which is a vital seismic method to obtain Earth inner structures. To raise the resolution of the imaging results, we need more seismic data for imaging. However, as the size of seismic data increases, the serial algorithm for NCF calculation becomes much more time-consuming. Thus, how to accelerate the NCF calculation becomes a key problem in ambient noise imaging. Based on the analysis of serial algorithm, we proposed a new parallel algorithm for NCF calculation using NVIDIA GPU platform. In addition, we improved reading and writing strategy to reduce I/O consumption. Experimental results on real seismic data show the effectiveness of our method. The parallel program achives about 1861 times speedup.**

***Index Terms*—Noise Cross-Correlation Function, Seismology, CUDA, GPU**

I. Introduction

Ambient noise imaging is an important method in seismology, which uses noise data recorded by large amounts of seismometers to obtain underground structures [1]. Calculating the noise cross-correlation functions (NCF) between seismic data is an essential part of ambient noise imaging [2]. Nowadays, more and more large-scale seismic observation arrays with hundreds or even thousands of seismometers are widely deployed, which operate for several months to several years, recording tremendous amounts of seismic data. This offers an opportunity to get much more accurate Earth's structural information than before [3]. However, since the data size is so large, it often takes weeks or even months to calculate the NCF, which then becomes a obstacle in scientific researches.

In recent years, advancing GPUs and parallel computing methods have been widely employed in scientific computing, which greatly accelerates computing. Meanwhile reducing I/O cost [4]–[6] also plays an important role in high performance scientific computing.

In this paper, we manage to accelerate the NCF calculation on NVIDIA GPU platform. In the following sections, we firstly introduce ambient noise imaging, NCF calculation and gpu computing briefly. Then, we analyze the serial algorithm of NCF calculation. Next, we propose our acceleration methods for the hotspot of original NCF algorithm. After that, We propose customary strategies to reduce I/O consumption. In the last section, we show the experimental results which exhibit the effectiveness of the proposed methods.

II. Background

A. Ambient noise imaging

Traditional seismological methods usually use the information carried by seismic phases, which are ”valid signals” in seismic data, to resolve structures in the Earth. And it has been supposed that noise accompanying the ”valid signals” is useless and decreases the accuracy of the result. To raise signal-noise ratio, seismologists apply various algorithms to suppress the noise in data [7].

In recent years, however, instead of removing the noise, more and more researches manage to analyze seismic noise to obtain new information about Earth's interior [8]. This gives birth to a new field named seismic ambient noise imaging.

The utilization of seismic noise is based on the coherence between different seismic records. By doing cross-correlation calculation of long-time seismic records, we can get NCF between them which characterizes their coherence. Then, the time derivatives of NCF are calculated to obtain empirical Green's function [9].

Green's function is the distribution of displacement field under the action of unit impulse. And through the convolution of Green's function and some seismic source's function, the distribution of displacement field under the action of that source can be determined. Thus, Green's function is very important in seismology since it contains the information of the medium inside the Earth. By calculating NCF and its time derivatives, we can acquire Green's function directly, and apply it in the subsequent seismological researches.

B. NCF calculation algorithm

Given two seismic signals $x(n)$, $y(n)$, the NCF between them is

$$r_{xy}(n) = \sum_{m=-\infty}^{+\infty} x(m)y(m+n) = x(-n) * y(n) \qquad (1)$$

which is exactly the convolution of $x(n)$'s reverse and $y(n)$.

Since the convolution operation is time consuming. In signal processing, the NCF calculation is often performed in frequency domain, where the convolution in time domain is replaced by more time-saving multiplication.

For seismic signal $x(n)$, fast Fourier transform (FFT) can be applied to transform it to its frequency domain counterpart

$$X(f) = FFT(x(n)) \tag{2}$$

The same operation is performed to $y(n)$ and we can get $Y(f)$.

Then, the NCF in the frequency domain is calculated by

$$R_{XY}(f) = \overline{X(f)}Y(f) \tag{3}$$

Finally, the time domain NCF can be calculated from $R_{XY}(f)$ by inverse fast Fourier transform (IFFT)

$$r_{xy}(n) = IFFT(R_{XY}(f)) \tag{4}$$

C. introduction to CUDA and CUFFT

CUDA is an extension of C/C++ language for parallel programming on NVIDIA GPUs. CUDA organizes threads hierarchically in blocks and grids. Each thread has its own local memory, and all threads in the same block can access the shared memory of that block. In addition, all threads can access data from the same global memory. Using global and shared memory, CUDA provides hierarchy parallelism between threads [10]. CUFFT is a library for computing FFT on NVIDIA GPUs. It employs Cooley-Tukey algorithm to accelerate the calculation of FFT [11].

III. Serial algorithm analysis

A. Algorithm

As Fig. 1 shows, the serial algorithm of NCF calculation consists of two parts: single station part and NCF part. The single station part takes every station's seismic data as input, and outputs the station's frequency spectrum data. The NCF part is calculated on each station pair's frequency spectrum and outputs NCF of each station pair.

In the single station part, after reading a station's seismic data, a sliding window segmentation is used to divide the original data into several segments. Each time domain segment is then transformed to frequency spectrum using FFT.

In the NCF part, each station pair's frequency spectrum are multiplied element-wise. Since the data has been segmented, the multiplication should be performed segment by segment. Then, IFFT is applied to each segment's results of multiplication, which produces NCF of each time segment. Next, NCF of all time segments are averaged. The output is the final NCF result between the input station pair.

B. Time complexity analysis

For N stations, the single station part of above algorithm is applied to each station's data, so this part will do N times. For each seismic signal, we divide it into M segments. Each segment's length is T_s, and neighboring segments have T_o overlap. Thus, the total length of each signal T is

$$T = MT_s + (M-1)T_o \tag{5}$$

For each signal, the segmentation and concatenation's time complexity is $O(T)$. The time complexity of M segments' FFT is $O(M \cdot T_s\log(T_s))$. Therefore, the time complexity of the single station part is

$$\begin{aligned} &O(N(T + MT_s\log(T_s))) \\ &= O(N(MT_s + (M-1)T_o + MT_s\log(T_s)) \end{aligned} \tag{6}$$

which is asymptotically equal to $O(NM(T_s + T_o + T_s\log(T_s))$

And the NCF part is done for each station pair. Thus, the calculation needs to be done $\frac{N(N-1)}{2}$ times. For M seismic spectrum segments of length F_s, where $F_s = \frac{T_s+1}{2}$, the time complexity of multiplication and averaging is $O(MT_s)$, and the time complexity of M segments' IFFT is $O(MT_s\log(T_s))$. Therefore, the time complexity of the NCF part is

$$O(\frac{N(N-1)}{2}(MT_s + MT_s\log(T_s))) \tag{7}$$

which is asymptotically equal to $O(N^2MT_s(1 + \log(T_s)))$.

In consequence, when N is very large, the time consumption of NCF part will be dominant in the total execution time. Because of that, in this paper, we will focus on accelerating the NCF part.

IV. CUDA Parallel algorithm

A. Reading data

In the NCF part of the serial algorithm, since every station participates in $N-1$ times NCF calculation of a certain day, one station's daily seismic data will be read $N-1$ times. Such redundant reading results in waste of time.

Like the NCF part, in parallel algorithm, we first read stations' 1-day frequency spectrum data into the memory. Based on the size of real seismic data, we have two strategies to read in data. The first one reads 1 station pair's data of all day in one execution and outputs 1 station pair's all-day NCF, while all station pairs being processed serially. The second one reads all stations' data of 1 day in one execution and outputs all station pairs' 1-day NCF, while all days being processed serially.

Compared to the former reading strategy, the latter one consumes much less time because it avoids redundant reading. For N stations, if we use the former strategy, once a station is involved in the NCF calculation, all-day data of that station will be read in. Since one station participates $N-1$ times NCF calculation, its all-day data will be read in for $N-1$ times, which is the same as the serial algorithm. However, if we use the latter strategy, all stations' 1-day data will only be used once when calculating NCF on that day. Thus, there is no redundant reading of one station's daily data. In other words, this reading strategy will reduce reading time to $\frac{1}{N-1}$ of former one. Consequently, we apply the latter reading strategy in this paper.

B. segment multiplication

After copying the frequency spectrum data into GPU memory, we will do segment multiplication for $\frac{N(N-1)}{2}$ station pairs. However, since the station number N is very large, we

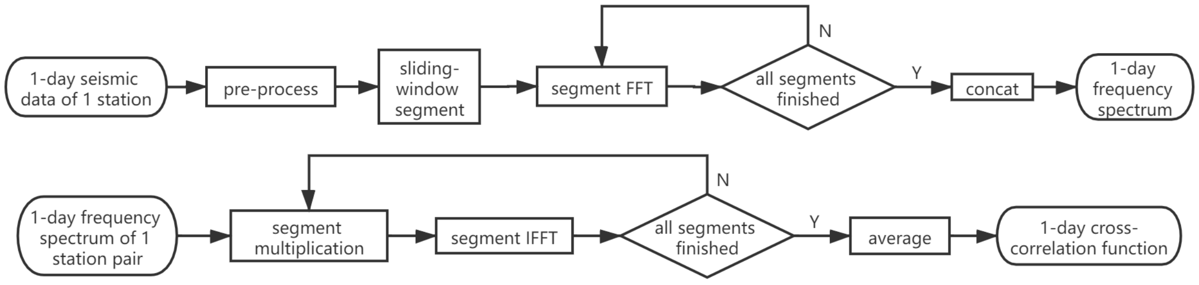

Fig. 1. Calculation flow of serial algorithm

can't do all times segment multiplication in parallel because of the limitation of GPU memory. Thus, at a time, we choose to do part of them in parallel.

Algorithm 1
Input: N; Output: $sp[\frac{N(N-1)}{2}, 2]$

```
1: $k \leftarrow 0$
2: for $i = 0$ to $N-2$ do
3:     for $j = i+1$ to $N-1$ do
4:         $sp[k, 0] \leftarrow i$
5:         $sp[k, 1] \leftarrow j$
6:         $k \leftarrow k + 1$
7:     end for
8: end for
9: return $sp$
```

As Algorithm 1 shows, before segment multiplication, we will create sp which is a list of all station pairs. Every time we do parallel segment multiplication, we will calculate for L station pairs, as shown in Algorithm 2 line 10 to line 20. The input of Algorithm 2 is N station's frequency spectrum data F, and the output of the segment multiplication is C_f, which is L station pairs' NCF of M segments in frequency domain. T_s is point number of FFT, whereas F_s is output length of FFT, dsatisfying $F_s = T_s/2 + 1$ when T_s is even.

C. segment IFFT

We do IFFT to the multiplication results C_f segment by segment with CUFFT library. After IFFT, we obtain C_t in Algorithm 2. Each segment of C_t is the NCF of corresponding station pair's segments.

D. averaging

The parallel averaging process is shown in Algorithm 2 line 23 to line 35. For the IFFT output C_t, we average corresponding data points of the segments in the same line and output lo rows of C. When loops of station pairs ends, we can obtain the whole C which is the NCF of $\dfrac{N(N-1)}{2}$ station pairs.

E. Writing results

In the case of serial calculation, only one station pair's NCF can be calculated after one execution of program. Thus, every station pair's NCF data is written as an individual file. This writing strategy decreases writing efficiency.

Different from the serial method, after our parallel program is executed once, we can obtain all station pairs' NCF results. Thus, instead of splitting the results into many small files, we write all the data into one file. This improvement might save some output time and alleviate the problem of high I/O consumption to some extent. We will analyze the effect of this improvement in the experiments section.

F. Time complexity analysis

For N stations, our method carried out $\lceil\dfrac{N(N-1)}{2L}\rceil$ loops. In each loop, NCF is calculated for L station pairs. The time complexity of segment multiplication is $O(1)$, since each element-wise multiplication can be assigned to different threads. The time complexity of segment IFFT is $O(T_s\log(T_s))$. For one station pair's M segments, the time complexity of averaging is $O(MT_s)$, and different station pairs' averaging is done in parallel. Therefore, the time complexity of our method is

$$\begin{aligned} &O(\lceil\frac{N(N-1)}{2L}\rceil(T_s\log(T_s) + MT_s)) \\ &= O(\lceil\frac{N(N-1)}{2L}\rceil T_s(\log(T_s) + M)) \end{aligned} \tag{8}$$

which is much smaller than serial NCF.

V. Experiments

A. lab environment

We use a data set containing 243 stations. Each station's frequency spectrum data has 23 segments, and each segment contains 4097 complex data points. In the writing step, 7201 data points at specified position out of all the 8192 data points are written to the output file.

The experiments are carried out on Tesla V100 GPU and Intel(R) Xeon(R) E5-2667 v3 CPU.

Algorithm 2

Input: $N, M, L, T_s, F[N, (\frac{T_s}{2}+1)\cdot M], sp[\frac{N(N-1)}{2}, 2]$, $gridDim, blockDim$;

Output: $C[\frac{N(N-1)}{2}, T_s]$

```
1: $F_s \leftarrow \frac{T_s}{2} + 1$
2: $n \leftarrow \lfloor \frac{N(N-1)}{2L} \rfloor + 1$
3: for $k = 0$ to $n-1$ do
4:     if $k < n-1$ then
5:         $lo \leftarrow L$
6:     else
7:         $lo \leftarrow \frac{N(N-1)}{2} - (n-1)\cdot L$
8:     end if
9:     $r \leftarrow k \cdot L$
10:    $y \leftarrow blockIdx.y$
11:    while $y < lo$ do
12:        $x \leftarrow blockIdx.x \cdot blockDim.x + threadIdx.x$
13:        while $x < F_s \cdot M$ do
14:            $S_A \leftarrow sp[r+y, 0]$
15:            $S_B \leftarrow sp[r+y, 1]$
16:            $C_f[y,x] \leftarrow ComplexMul(ComplexConj($
                   $F[S_A, x]), F[S_B, x])$
17:            $x \leftarrow x + gridDim.x \cdot blockDim.x$
18:        end while
19:        $y \leftarrow y + gridDim.y$
20:    end while
21:    $C_t \leftarrow segment\_IFFT(C_f)$
22:    $y \leftarrow blockIdx.y$
23:    while $y < lo$ do
24:        $x \leftarrow blockIdx.x \cdot blockDim.x + threadIdx.x$
25:        while $x < T_s$ do
26:            $sum \leftarrow 0.0$
27:            for $i = 0$ to $M-1$ do
28:                $sum \leftarrow sum + C_t[y, x + i \cdot T_s]$
29:            end for
30:            $C[r+y, x] \leftarrow \frac{sum}{M}$
31:            $x \leftarrow x + gridDim.x \cdot blockDim.x$
32:        end while
33:        $y \leftarrow y + gridDim.y$
34:    end while
35: end for
36: return $C$
```

B. experimental results and analysis

The experimental results is shown in TABLE I. Since parallel 1 and parallel 2 only differs in output strategy, so they have the same input time and calculation time.

In the results, we can see that parallel algorithm accelerate the calculation by 1861 times. In addition, owing to the avoidance of redundant reading, parallel algorithm reduces much input time, and the input acceleration rate 214 is approximately equal to $N-1$, which agrees to the theoretical analysis.

As for the difference of two parallel methods, parallel 2 reduces more output time than parallel 1. In terms of the total time, parallel 1's acceleration rate is 737 and parallel 2's is 1214. Thus, we can conclude that writing all NCF data to one file can accelerate the output process.

TABLE I
EXPERIMENTAL RESULTS

Methods	serial	parallel 1	parallel 2
Total time(s)	2534.650	3.437	2.088
Input time(s)	55.895	0.261	0.261
Calculation time(s)	2473.392	1.329	1.329
Output time(s)	5.363	1.846	0.498
Total acc. rate	-	737	1214
Input acc. rate	-	214	214
Calculation acc. rate	-	1861	1861
Output acc. rate	-	2.9	10.8

parallel 1: every NCF data is written to an individual file.
parallel 2: all NCF data is written to one file.

VI. Conclusion

In this paper, we manage to realize the acceleration of NCF calculation. After analyzing the serial algorithm for NCF calculation, we focus on the acceleration of the algorithm's NCF part. We propose a new parallel method using CUDA and CUFFT and alter I/O strategies. In the experiments, the total acceleration ratio of our algorithm is 1214. More importantly, our algorithm accelerates the calculation part by 1861 times. Such results verifies our method's good acceleration performance on NCF calculation.

Acknowledgment

This work is supported by National Natural Science Foundation of China (No.61772485).

References

[1] Weaver, Richard, and L. "Information from Seismic Noise. " Science (2005).
[2] N. Li, W. Wang, B. Wang, "Speeding the Nine-component Cross Correlation Function Calculation Using Cloud-computing and Its Application on the Dataset of China Array-NE Tibet." Earthquake Research in China, 2018
[3] Lin, F. C. , et al. "Complex and variable crustal and uppermost mantle seismic anisotropy in the western United States." Nature Geoscience 4.1(2011):55-61.
[4] Zhang, L., Qiu, M., Tseng, WC. et al. Variable Partitioning and Scheduling for MPSoC with Virtually Shared Scratch Pad Memory. J Sign Process Syst Sign Image Video Technol 58, 247–265 (2010). https://doi.org/10.1007/s11265-009-0362-3
[5] Y. Guo, Q. Zhuge, J. Hu, M. Qiu and E. H. -. Sha, "Optimal Data Allocation for Scratch-Pad Memory on Embedded Multi-core Systems," 2011 International Conference on Parallel Processing, 2011, pp. 464-471.
[6] M. Qiu, Z. Chen and M. Liu, "Low-Power Low-Latency Data Allocation for Hybrid Scratch-Pad Memory," in IEEE Embedded Systems Letters, vol. 6, no. 4, pp. 69-72, Dec. 2014.
[7] Rost, and Sebastian. "ARRAY SEISMOLOGY: METHODS AND APPLICATIONS." Reviews of Geophysics 40.3(2002):2-1-2-27.
[8] Yao, H. , D. Van , and M. D. Hoop . "Surface-wave array tomography in SE Tibet from ambient seismic noise and two-station analysis – I. Phase velocity maps." Geophysical Journal International (2006).
[9] Wapenaar, K. . "Retrieving the Elastodynamic Green's Function of an Arbitrary Inhomogeneous Medium by Cross Correlation." Physical Review Letters 93.25(2005):254301.
[10] Nickolls, John R , et al. "Scalable parallel programming with CUDA." Queue (2008).
[11] cuFFT :: CUDA Toolkit Documentation. https://docs.NVIDIA.com/cuda/cufft/, Dec. 2019