Parallel Computing Method for HRV Time-Domain Based on GPU

Jie Wang, Weihao Chen, and Gang Hou(✉)

School of Software Technology, Dalian University of Technology,
Economic and Technological Development Zone, Dalian 116600, China
{Wangjie1003, Endeavour35, hg.dut}@163.com

Abstract. HRV (Heart rate variability, which has a function of prediction for cardiovascular disease) contains a wealth of medical information, rapid extraction and procession of these signals will bring an important meaning for the prevention of heart diseases. Physionet open source project provides a good platform for the research and development of HRV, which also provides demonstration tools for the calculation of HRV. The characteristics of medical signal are real-time and have large volume of data. Conventional serial methods are difficult to meet the requirements of biomedicine, and the parallel method based on multi-core CPU is larger communication overhead. In this paper, we designed some parallel algorithms for the calculation of HRV in time-domain based on the strategy of parallel reduction, compared and analyzed the various optimization methods, and received the highest 38 times speedup compared with serial method.

Keywords: GPU · HRV · Parallel · Reduction · Time-domain

1 Introduction

Cardiovascular disease is one of the major diseases that threaten human life [1], which has features of high incidence, high morbidity, and high mortality. World Health Organization statistics shows that about 17 million people worldwide died of all types of cardiovascular disease each year. For the diagnosis of heart diseases, treatment and prevention is still the huge challenge of medical profession.

ECG (Electrocardiogram) [2] diagnosis is a very important way for cardiovascular disease diagnosis, which has a high value [3, 4] for the identification of all types of arrhythmias, suggesting atrial, ventricular hypertrophy, the diagnosis of myocardial infarction, myocarditis and other diseases. With the deepening of HRV study, physiological and pathological information that it contains will be further revealed, which will make HRV contains more space and value of application. Currently, heart rate variability analysis method [5, 6] contains time-domain analysis, frequency-domain analysis, time-frequency analysis and nonlinear analysis. Time-domain analysis is the easiest way of measuring heart rate variability signal, which is a method of statistical analysis. Researchers discrete RR interval changes based on the method of trend analysis by statistics. Analysis in time-domain is simple, intuitive and easy for the acceptance of the clinicians, and has accumulated a lot of experience.

Medicine signal characteristics: informative, randomness and noise background are strong, which makes it require long time for the measurement of biomedical signal. HRV time-domain analysis in general should be a long-term of at least 24 h, especially for acute myocardial infarction (AMI) prognosis, so it is extremely important for the rapid processing of mass of HRV data. GPU is widely used in biomedicine [7–9], and the calculation methods used for HRV and optimization are very necessary.

2 The Current Status of Research

2.1 Open Source Project of PhysioNet

In order to promote the exchange and cooperation in medical field, the US National Institutes of Health (NIH) funded, Massachusetts Institute of Technology (MIT) jointly BethIsrael Deaconess Medical Center in Boston, etc. established a web-based complex physiological and biomedical signal resources website PhysioNet (www.physionet.org) since 1999, to facilitate the exchange of research data and analysis software, and encourage extensive collaboration between researchers. Physionet medical researchers can apply them to their own research, to validate and assess the range of practical algorithms.

A signal data of PhysioBank database consists of multiple files, a major component consists of three parts: the header file, data file and comment file, while the header file (.hea) is essential document among them. As shown in Fig. 1, website also provides analytical tools that can be used to calculate various HRV parameters. The open-source project provides a good platform for the cooperation and exchange of biomedical science development.

2.2 Prospects of GPU in Biomedicine

Speed and accuracy [10] are the key index of signal evaluation, which is more and more important. With the increasing amount of data to be processed, especially for situation that requires 24-hour monitoring and real-time processing, the execution time will increase as the amount of data gradually extend.

Applications of GPU in biomedicine are widespread. Saha and Desai [11] made a real-time analysis of huge amount of data generated by multichannel body sensor

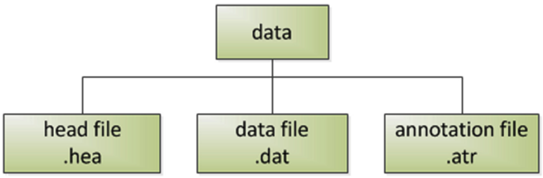

Fig. 1. The data structure of signals

networks such as Electroencephalogram (EEG), Electrocardiogram (ECG), Electromyography (EMG) and Functional-MRI (fMRI), which is critical for the deployment of these technologies. Processing these multichannel data sets using complex signal processing algorithms such as Grassberger and Procaccia (GP) algorithm to compute Correlation Dimension (D2) is resource intensive. Using massively parallel computing infrastructure comprising of Graphical Processing Units (GPU) and Multicore Processing nodes for parallelizing such huge data dependent problems is an apt use of the resource (Tables 1 and 2).

Table 1. The result comparison A

ECG sample	Computing infrastructure	No. of samples	Time taken for all samples	Speedup for complete data
MIT BIH 101	Sequential C Code	5120	60.4998	1
		10240	283.752	1
	MPI/Thread Hybrid	5120	3.32361	18.20303826
		10240	17.8462	15.89985543
	GPU Implementation#1	5120	14.837	4.077630249
		10240	91.12	3.11404741
	GPU Implementation#2	5120	6.28	9.633726115
		10240	96.25	2.948072727

Table 2. The result comparison B

ECG sample	Computing infrastructure	No. of samples	Time taken for one iteration of "r"	Speedup for one iteration
MIT BIH 101	Sequential C Code	5120	0.43	1
		10240	1.59	1
	MPI/Thread Hybrid	5120	0.42	1.023809524
		10240	1.49	1.067114094
	GPU Implementation#1	5120	0.0086	50
		10240	0.045	35.3333333
	GPU Implementation#2	5120	0.0086	50
		10240	0.045	35.3333333

The authors took two different databases for this experiment, and tested the algorithms of Sequential C Code, MPI/Thread Hybrid, GPU implementation#1, GPU implementation#2. The two algorithms of GPU can get ideal speedup in one iteration, and have an obvious speedup when compared with algorithm of sequential C code.

The implementation results of traditional serial method is slow, which is hard to deal with HRV parameters for the calculation of large amounts of data, and is difficult

Table 3. Time-domain parameters formula

$$\text{AVNN} = \frac{\sum_{i=1}^{N} R_i}{N} \tag{1}$$

$$\text{SDNN} = \sqrt{\frac{\sum_{i=1}^{N}(R_i - \bar{R})^2}{N}} \tag{2}$$

$$\text{SDANN} = \sqrt{\frac{\sum_{i=1}^{N}(aR_i - \overline{aR})^2}{N}} \tag{3}$$

$$\text{SDNNIDX} = \frac{SDNNi}{Num} \tag{4}$$

$$\text{rMSSD}* = \sqrt{\frac{\sum_{i=1}^{N}(R_i - R_{i-1})^2}{N}} \tag{5}$$

$$\text{pNN50} = \frac{NN_{>50}}{TotalNN} \times 100\% \tag{6}$$

to meet the medical requirements for real-time detection. Aiming at the characteristics of HRV parameters in time-domain, a parallel strategy of reduction will be applied. The experiment showed that it could achieve good acceleration when compared with traditional serial algorithm (Table 3).

3 Serial Analysis of Time-Domain of HRV

Parameters in time-domain of HRV are calculated above, the calculation processes based on the above formula is shown as follows:

Step 1: Get files of PhysioBank database of corresponding disease;
Step 2: Call QRS detection program [12];
Step 3: Generate document of RR interval sequence;
Step 4: Read file and compute time-domain parameters;
Step 5: Output the results.

According to the calculation process, it is easy to write the appropriate serial computing program. But there are lots of plus operations in the calculation of time-domain, and it is difficult to meet the requirements of real time ECG analysis when operating a large amount of data. So optimization of this part is crucial.

4 Parallel Calculation of Time-Domain HRV

4.1 Parallel Optimization Principle

In the calculation of time-domain HRV, a number of data encryption and storage will be performed. Speed and accuracy are the key indicators to evaluate signal processing algorithm, especially in the field of biomedical.

Parallel reduction [13] is a common and important optimization method, which can be applied to more than one thread blocks and guarantee GPU processors in a busy state. By dividing calculations into multiple kernels, global synchronization operation will be avoided, communication overhead will be reduced too (Fig. 2).

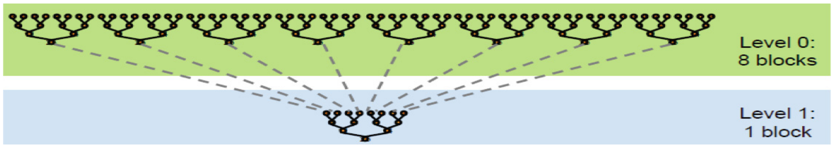

Fig. 2. Parallel reduction method

4.2 Strategy of Reduction

These are applets of the calculation of HRV in time-domain. THREAD_NUM is 512, which is the block Dimension. At first, the data is stored in an array vec1 in global memory. When the applet starts, data will be copied to the array vec, which is allotted by every block. At the end of every applets, the data in the first thread of each block will be copied to nnCounter that is allotted in global memory.

Sequential Addressing.

```
__shared__ int vec[THREAD_NUM];//call for the shared momory for 512
vec[index] = vec1[id];      //threads, and copy the data form global
  __syncthreads();          // memory to shared memory
  for(int i=blockDim.x/2;i>0;i/=2){
    if(index < i)
      vec[index]+= vec[index+i];
        __syncthreads();
  }
  if(threadIdx.x == 0)      //add the sum of blocks to the first thread
    nnCounter[blockIdx.x] = vec[0];
```

Shared memory is second only to register, and has a faster execution. In this part of optimization, every vector will be stored in shared memory of the block. By the judgment of (index < i) for each thread, bank conflict will be avoided by the thread discontinuously.

First Add During Global Load.

```
__shared__ float vec[THREAD_NUM];
int id = blockIdx.x * (blockDim.x*2) + threadIdx.x;
int index = threadIdx.x;
vec[index] = vec1[id]+vec1[id+blockDim.x];//two blocks
__syncthreads();                          //add together
for(int i=blockDim.x/2;i>0;i/=2){
  if(index < i)
    vec[index]+= vec[index+i];
      __syncthreads();
}
if(threadIdx.x == 0)
  vec0[blockIdx.x] = (int)vec[0];
```

Drawbacks of the above optimization algorithm are that half threads will be idle during each cycle. In order to reduce idle threads as much as possible, addition operations will be done when accessing global memory, so each block can use the same threads to complete twice operations of statue summing elements.

Unroll The Last Warp. The execution unit of GPU thread is warp, each warp thread within no synchronization. So when there are only 32 threads of a warp need to added, other warps will execute null branch. Warp synchronized will not be needed because of the sequential consistency, and when blockDim.x ≥ 32, operations will be operated based on reduce2.

```
__device__ void warpReduce1(volatile int* vec,int index)
{
  vec[index]+=vec[index+32];//unroll the warp
  vec[index]+=vec[index+16];
  vec[index]+=vec[index+8];
  vec[index]+=vec[index+4];
  vec[index]+=vec[index+2];
  vec[index]+=vec[index+1];
}
```

Complete Unrolling. CPU execution cycle is an inefficient recycling portion, which can be fully extended. When the number of threads in each block is n-th power of 2, the efficiency of the code will be higher.

```
        vec[index] = vec1[id]+vec1[id+blockDim.x];
        __syncthreads();
        if(blockDim.x>=512){
          if(index<256){
            vec[index]+=vec[index+256];}
        __syncthreads();}
        if(blockDim.x>=256){
          if(index<128){
            vec[index]+=vec[index+128];}
        __syncthreads();}
        if(blockDim.x>=128){
          if(index<64){
            vec[index]+=vec[index+64];}
        __syncthreads();}
        if(index<32)
          warpReduce1(vec,index);
        if(threadIdx.x == 0)
          vec0[blockIdx.x] = vec[0];
```

Multiple Adds/Threads.

```
    __shared__ int vec[THREAD_NUM];
    int id = blockIdx.x * (blockDim.x*2) + threadIdx.x;
    int index = threadIdx.x;
    int gridSize = blockDim.x*2*gridDim.x;
    vec[index] = 0;
    while(id < MAX_PEAK) // MAX_PEAK is all threads number
    {
      vec[index] = vec1[id]+vec1[id+blockDim.x];
      id += gridSize;
    }
    __syncthreads();
```

Loop reading of the global memory and copying data to shared memory. In this cycle, vast majority of warps are able to maintain full load and is possible to improve efficiency.

5 Implementation of the Results and Analysis of HRV

5.1 Introduction of the Platform

The testing environment of parallel reduction was run in Linux, GPU is GeForce GTX460 of NVIDIA. GTX460 computing power of 2.1, memory 1G, memory interface

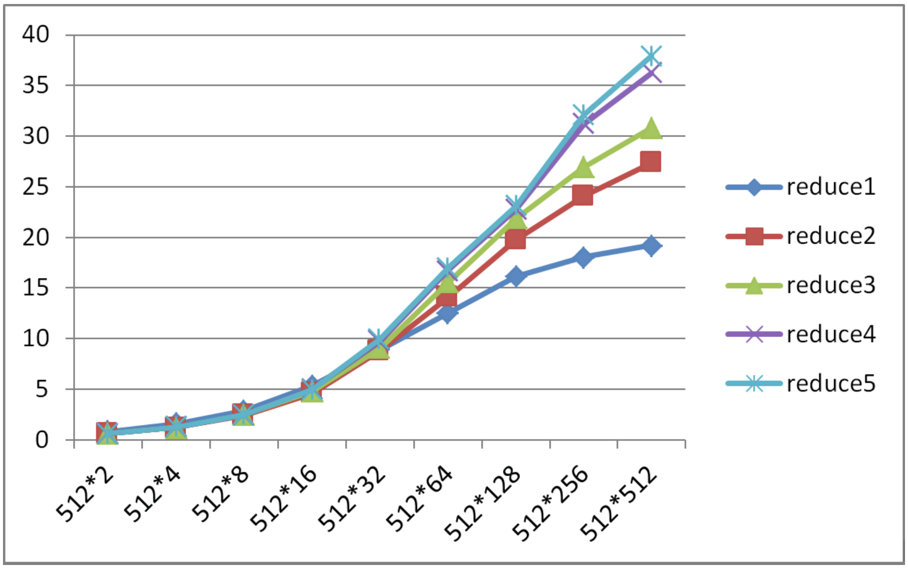

Fig. 3. According to the table to draw the appropriate line chart is as above.

256-bit, memory frequency 1848.00 MHz, a total of 336 CUDA [14, 15] Cores, GPU clocked at 1.40 GHz. Because GTX460 uses the latest Fermi architecture, so the number of threads is 1024. In order to make the code more portable, we allocate the maximum 512 threads per block (Fig. 3).

5.2 Analysis of the Results

Analysis of different sizes for each algorithm speedup situation Tables 4 and 5.

When the data size is less than 512 * 4, the result of GPU parallel optimization acceleration is not ideal, and the difference of the acceleration effect between various optimization methods is not great. So the acceleration effect of various optimization strategies is not ideal when the data set is small, and the difference between them is small too.

When the data size increases to 512 * 64, the acceleration effect gap between them is able to reflect. The strategy of first add of reduce2 has greatly improved when

Table 4. Points table with speedup A

	512*2	512*4	512*8	512*16	512*32
reduce1	0.79	1.58	2.87	5.38	8.92
reduce2	0.62	1.22	2.42	4.55	8.78
reduce3	0.62	1.22	2.47	4.78	9.03
reduce4	0.64	1.28	2.5	5.03	9.74
reduce5	0.64	1.27	2.5	4.94	10.02

Table 5. Points table with speedup B

	512*64	512*128	512*256	512*512
reduce1	12.56	16.15	18.06	19.2
reduce2	14.1	19.77	24.08	27.35
reduce3	15.42	21.88	26.97	30.81
reduce4	16.77	22.75	31.22	36.23
reduce5	17.2	23.11	32.14	37.91

compared with strategy that used shared memory only, and the accelerating effect will be improved further when unrolling the last warp or unroll them completely.

From the above figure, we can also analyze that with the node-by-fold increasing in the size, speed node ratio does not increase exponentially, but is gradually leveled off. From the trend of reduce1, we can see that each acceleration policies have the appropriate stability threshold. To make the algorithm accelerate further, we can use the new architectures such as Kepler hardware, or optimize the transmission bandwidth.

5.3 The Complexity of the Parallel Reduction

Parallel reduction will perform Log(N) steps, each step S does $N/2^2$ independent ops, step complexity is O(logN); For $N = 2^D$ perform $\sum_{S\in[1...D]} 2^{D-S} = N - 1$ operations, work complexity is O(N). It is work-efficient, and it does not perform more operations than a sequential algorithm; With P threads physically in parallel (P processors), time complexity is O(N/P + logN), in a thread block, N = P, so O(logN).

6 Conclusion and Future Work

HRV as an important index for cardiovascular and cerebrovascular disease, quick and accurate are crucial. This article designed and implemented the calculation of time-domain of HRV parameters, which received nearly 35 times speedup. For different ECG database, according to the characteristics of each parameter calculation, using the appropriate strategy of parallelization can bring great improvement for the calculation of the various parameters.

GPU distributed computing model promotes the development of the top supercomputers. MPI enables CUDA extend to compute thousands of nodes. At the same time, commercial-grade GPU clusters and cloud computing platform provides a full range of resources for users and organizations. We designed some HRV parallel computing methods based on GPU, and analysis the speedup of each algorithm. With the rapid increase of medical data, combining with GPU and distributed computing, taking advantage of GPU cluster analysis process will greatly improve the calculation of HRV and other mass medical data.

Acknowledgments. This paper was supported by National Nature Science Foundation of China (No. 61472100) and Fundamental Research Funds for the Central Universities, China (No. DUT14QY32).

References

1. Thayer, J.F., Yamamoto, S.S., Brosschot, J.F.: The relationship of autonomic imbalance, heart rate variability and cardiovascular disease risk factors. Int. J. Cardiol. **141**, 122–131 (2010). Elseiver
2. Bansal, D., Singh, V.R.: Algorithm for online detection of HRV from coherent ECG and carotid pulse wave. Int. J. Biomed. Eng. Technol. (IJBET) **14**, 333–343 (2014). Inderscience Publishers Ltd
3. Ferrari, E., Imbert, A., Chevalier, T., et al.: The ECG in pulmonary embolism: predictive value of negative T waves in precordial leads—80 case reports. CHEST J. **113**, 537–543 (1997). American College of Chest Physicians
4. Zhou, Q.: ECG heart beat modeling and analysis to identify. PhD thesis of Zhejiang University, Biomedical Engineering and Instrument Science, Zhejiang University, Zhejiang, pp. 5–10 (2004)
5. Mohan, A., et al.: Design and development of a heart rate variability analyzer. J. Med. Syst. **36**, 1365–1371 (2012). Springer
6. Jeppesen, J., Fuglsang-Frederiksen, A., Brugada, R., et al.: Heart rate variability analysis indicates preictal parasympathetic overdrive preceding seizure-induced cardiac dysrhythmias leading to sudden unexpected death in a patient with epilepsy. Epilepsia **55**, e67–e71 (2014). Wiley Online Library
7. Konstantinidis, E.I., et al.: Accelerating biomedical signal processing algorithms with parallel programming on graphic processor units. In: ITAB2009. IEEE, pp. 1–4 (2009)
8. Freiberger, M., et al.: The agile library for biomedical image reconstruction using GPU acceleration. Comput. Sci. Eng. **15**, 34–44 (2013). AIP Publishing
9. Costa, C.M., Haase, G., Liebmann, M., Neic, A., Plank, G.: Stepping into fully GPU accelerated biomedical applications. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2013. LNCS, vol. 8353, pp. 3–14. Springer, Heidelberg (2014)
10. Konstantinidis, E.I., et al.: Accelerating biomedical signal processing algorithms with parallel programming on graphic processor units. In: Information Technology and Application in Biomedicine, ITAB. IEEE, pp. 1–4 (2009)
11. Saha, D.P., Desai, A.R.: Performance analysis of computing multichannel correlation dimension (D2) on multicore system and GPU. Researchgate.net
12. Tarvainen, M.P., et al.: Kubios HRV – heart rate variability analysis software. Comput. Methods Programs Biomed. **113**, 210–220 (2014). Elseiver
13. Harris, M.: Optimizing parallel reduction in CUDA. Presentation packaged with CUDA Toolkit, NVIDIA Corporation (2007)
14. NVIDIA Corporation, NVIDIA CUDA Programming Guide (2009)
15. http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf