International Conference on Applied Internet and Information Technologies, 2016
DOI:10.20544/AIIT2016.35

CUDA DSP Filter for ECG Signals

Ervin Domazet, Marjan Gusev, and Sasko Ristov

Ss. Cyril and Methodius University, Faculty of Computer Science and Engineering
1000 Skopje, Macedonia
ervin domazet@hotmail.com,{marjan.gushev, sashko.ristov}@finki.ukim.mk

Abstract. Real time processing is very important and critical for analysis of ECG signals. Prior to each processing, the signal needs to be filtered to enable feature extraction and further analysis. In case of building a data processing center that analyzes thousands of connected ECG sensors, one expects that the signal processing needs to be done very fast. In this paper, we focus on parallelizing the sequential DSP filter for processing of heart signals on GPU cores. We set a hypothesis that a GPU version is much faster than the CPU version. In this paper we have provided several experiments to test the validity of this hypothesis and to compare the performance of the parallelized GPU code with the sequential code. Assuming that the hypothesis is valid, we would also like to find what is the optimal size of the threads per block to obtain the maximum speedup. Our analysis shows that parallelized GPU code achieves linear speedups and is much more efficient than the classical single processor sequential processing.

Keywords: DSP, ECG, Heart Signal, Parallelization, CUDA, GPGPU.

1. Introduction

Due to low electrical levels, ECG signals are accompanied by noise, stemming from different sources, such as electrical switching power, radio waves and internal human breathing physical movement. Data preprocessing is essential for doing precise analysis and interpretation.

Digital Signal Processing (DSP) enables analysis of the ECG signal, with the intention to extract and interpret hidden data. Filters are among the most important tools of the DSP area, and are used to eliminate noise from the ECG signal. In some special circumstances, the real time processing of the ECG signal can even save lives. Sequential algorithms are insufficient of processing ECG signals in real time, especially when thousands of signals from remote wearable ECG sensors are reaching to the cloud data server.

In our previous research [4] we focussed on parallelizing DSP filters on Maxeler dataflow cores. We obtained relatively good results, with linear speedups proportional to the kernel length.

In this research, we focus on parallelizing the sequential DSP filter for processing of heart signals on graphics processing unit (GPU) cores. Implementation of FIR low pass, high bass and bandpass filters are used in our experiments for variable noise components of the ECG signal. We consider using CUDA library, which is a parallel

computing platform and programming model invented by NVIDIA. CUDA-enabled GPUs makes it possible for developers to use GPUs for general processing.

The hypothesis set in this paper is to confirm if the GPU DSP filter processing of the ECG signals is faster than the CPU processing. To test the hypothesis we will measure the speedup of the obtained GPU solution over the serial solution. Further one, our research question is to determine the optimal size of threads per block to obtain the maximum speedup.

The paper is organized as follows. The background about ECG signals and CUDA advantages is presented in Section 2. Section 3 presents DSP filters and convolution operators. An explanation of the parallelization method and the obtained CUDA code of the DSP filter is presented in Section 4. Obtained results of measured processing time and performance analysis of the parallelized code versus the original sequential code are presented in Section 5. Finally, the paper is concluded with a discussion and future work in Section 7.

2. Background

ECG holds significant information related to the cardiovascular condition of the human. This makes the analysis and interpretation significant for a healthier life. Figure 1 shows the general method for processing and analyzing ECG signals [6].

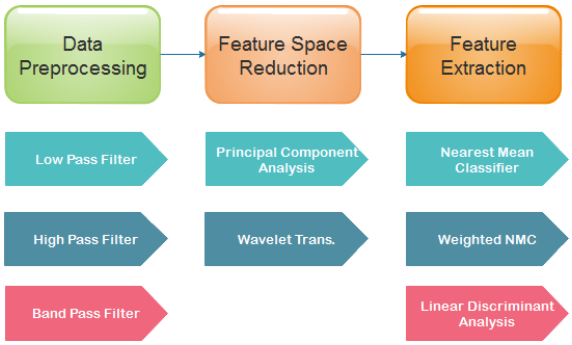


Fig.1. Methodology for ECG signal processing.

DSP involves a variety of tools aimed for signal analysis [3],[8], and DSP filters are considered as very important tools, useful for noise elimination. After noise elimination, features can be extracted for analyzing the complex heart condition. ECG features can vary from detecting hidden information and subtle deviation of the heart rhythm to alternating changes of the wave amplitude [2].

CUDA, or Compute Unified Device Architecture, is NVIDIA's technology which ensures a significant increase in software performance by using the GPU power [5,7]. The difference between a CPU and GPU is to compare how they process tasks. CPU consists of a few cores optimized for sequential serial processing, while the GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously [1].

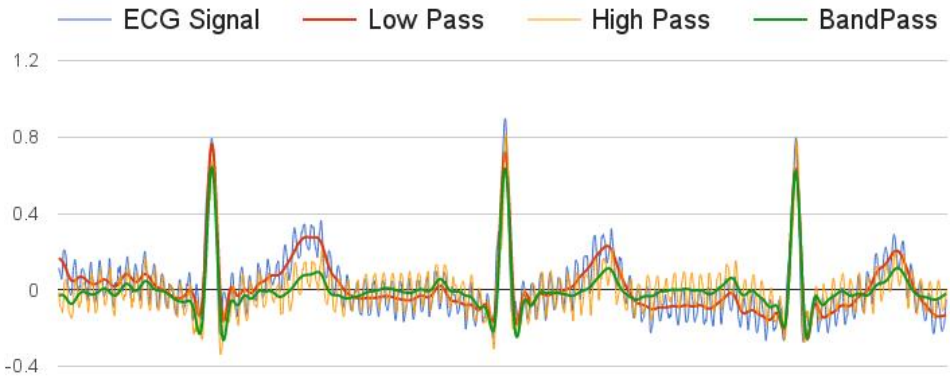


Fig.2. A segment of an ECG signal with several QRS complexes, filtered with a low pass filter of 30Hz, high pass filter of 0.5Hz and a band pass filter between 0.5Hz and 30Hz.

3. Digital Signal Processing

Digital Signal Processing (DSP) is considered as a set of tools for processing signals. They vary from filtering, measuring to producing or compressing analog signals. The power of this field in computer science dramatically increased during last decades [8]. Most complex systems use revolutionary DSP methods.

The focus of this research is on data preprocessing phase of ECG signal analysis. This phase has the intention to eliminate the noise of the signal. In this respect, DSP filters are used as tools of the first phase.

The most widely used method for a DSP filter is the *convolution*, defined to generate a new *output stream* as a mathematical operation of the *input stream* and *impulse response* The impulse response of a filter is usually denoted as a *filter kernel* using the weight coefficients $h(i)$. In case of an Infinite Impulse Response (IIR) Filter, $i \in \{-\infty,+\infty\}$. while, in the Finite Impulse Response (FIR) filters, $i \in \{0,\ldots,M-1\}$, where M is the filter length.

Denote by $x(i)$ the elements of the input stream and by $y(i)$ the elements of the output stream for $i = 0,\ldots$. The convolution is a mathematical operation calculated by (1).

$$y(i) = \sum_{j=0}^{M-1} h(j)x(i-j) \tag{1}$$

Three classic filters are used in this research to eliminate the noise: low-pass, highpass and band-pass filters. Figure 3 reveals simple frequency responses for low-pass, band-pass and high-pass filters respectively.

Low-pass filters weaken all the frequencies above the cutoff frequency, known as a *stopband*, while passing all frequencies below the *passband* [8]. All samples of the output stream are in fact a weighted average of the input with the adjacent points of the low pass filter.

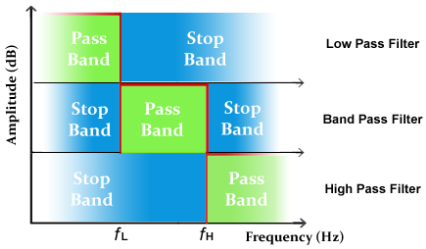


Fig.3. Frequency responses of the low-pass, band-pass and high-pass filters.

The high-pass filter is functioning completely opposite of the low-pass filter. The main effect of the filter is to weaken the frequencies below the *cutoff frequency* whereas passing all frequencies above the cutoff frequency.

A band-pass filter is a combination of the high-pass and low-pass filters. It passes a predefined range of frequencies and rejects the frequencies of the remaining region.

Figure 2 presents the effect of applying a low pass filter with a 30 Hz cut-off frequency on the ECG signal. It can be noted that 50Hz noise is eliminated. The effect of applying a high pass filtering with a cut-off frequency of 0.5 Hz is the elimination of the baseline drift, caused by breathing and other physical movements. The band pass filter by its nature is a combination of low pass and high pass filters. It eliminates both the baseline drift, caused by breathing and other physical movements and also the 50Hz noise caused by the electricity.

4. Parallelization for GPU Computing

The algorithm presented in Figure 4(a) is a sequential version of a convolution of a one dimensional input with a corresponding kernel. The complexity of the algorithm depends on the input and the kernel stream length, i.e $O(nm)$. When run on a CPU, the flow is sequential, meaning that the inner loop length depends on the kernel size.

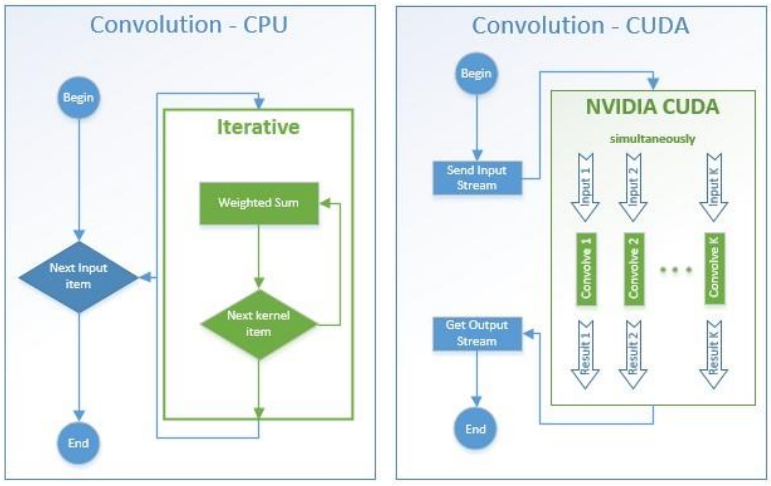
GPU computing is a rather different computing paradigm when compared to the traditional CPUs. GPUs have a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. Using this key feature, we have achieved a row based parallelization of convolution computation for the ECG input signal.

The sequential loop iterating the input signal is massively parallelized by synchronously utilizing thousands of GPU cores. Proposed solution is visualized in Figure 4(b).

General purpose GPU programing on NVIDIA CUDA enables modifiable execution plans, which are specified by blocks and grids. Block is considered as a group of threads. Its important to note that threads inside the blocks can be synchronized. Threads across same block can communicate via a shared memory

dedicated to the block. Besides, a grid is a group of blocks. CUDA does not provide means for synchronization between blocks. Moreover, a blocks inside a grid can communicate via the global memory.

This row version of the algorithm uses threads which size is proportional to the filter kernel. Whenever the threads are executed on a streaming multiprocessor of a GPU, a



(a) Sequential Solution (b) Parallel GPU Solution

Fig.4. Flow of sequential and GPU computing filtering algorithm.

number of threads per block (TPB) is defined. We plan to test several values of TPB and determine its optimal size.

Note that this solution does not include any optimization of the CUDA code, and we are aware of a lot of concurrent memory reads. Still, the distribution of the workload on the GPU cores, makes the advantage of using 1536 cores, even the system clock is much lower.

5. Experimental results

This section describes the conducted experiments and presents the obtained results.

5.1 Testing environment

The sequential and parallelized code are tested on an Amazon G2 2xlarge instance. It consists of a 8-core Intel(R) Xeon(R) CPU E5-2670 2.60GHz 64-bit system with a 15GB of memory. Additionally, the parallelized code is tested on a NVIDIA GRID (Kepler GK104) device, having 1,536 CUDA cores with 800Mhz system clock and 4GB RAM.

In this research we have experimented with the Hamming window and the Blackman window with length of 100 and 200 elements to obtain relatively good results.

5.2 Functional Verification

Several experiments are conducted to verify if the functional characteristics of the sequential and parallel algorithms are identical The input was a short stream of 1000 samples of an ECG signal with all characteristic P, Q, R, S and T waves, as presented in Figure 2.

We have verified both the sequential and parallelized CUDA and sequential solutions obtain identical results.

5.3 Test data

The measured parameters in the experiments are the time required to process the sequential algorithm T_s, and the time required to process the parallel algorithm with p cores, denoted by T_p.

The speed-up is calculated by (2) as a ratio of the measured times for execution of the sequential and parallel algorithms.

$$S_P = \frac{T_s}{T_p} \tag{2}$$

5.4 Results

Note that for each input signal, the speedup values are calculated by using T_s and T_p values for the same input configuration. Hence, the main reason for using these values is to compare the performance of the sequential code on CPU and the parallelized code on CUDA enabled GPU.

A total of 3 experiments were conducted, where input length varies from 10.000 to 500.000 in steps of 10.000 samples. The tested kernel length was 100, 500, 2000, 5000 and 7500.

Table 1 presents the running times of the sequential and parallel algorithms tested in our experiments only for selected values.

One can observe that the speedup increases as the input length is increased as it was expected. Moreover, the speedup is increasing slightly as the kernel length increases, due to a more efficient utilization of GPU cores without extra costs for allocating and dealocating a core.

Table 1. Speed-up analysis as the input size and kernel increase.

No	ECG Signal Length	Kernel Length	CPU Running Time (ms)	GPU Running Time(ms)	Speedup
1	10000	100	5.3	0.6	9.1
	50000		26	1	27.8
	100000		54	2	30.5
	500000		266	6	42.8
2	10000	500	27	1	40.5
	50000		132	2	60.7
	100000		264	4	66.9
	500000		1323	16	80.9
3	10000	2500	113	1	79.5
	50000		670	8	85.8
	100000		1275	14	92.9
	500000		5684	57	99.7

6. Discussion

This section evaluates and discusses the obtained results, and also provides an analysis of the TPB number to determine an optimal configuration of the algorithm.

Speedup of CUDA GPU vs CPU Solution The conducted experiments are tested for various input and kernel length. Figure 5 presents the speedup of the CUDA algorithm compared to the sequential version, where the input length varies from 10.000 to 500.000 samples and filter kernel lengths of 100, 1.000 and 2.500, with TPB value of 1024.

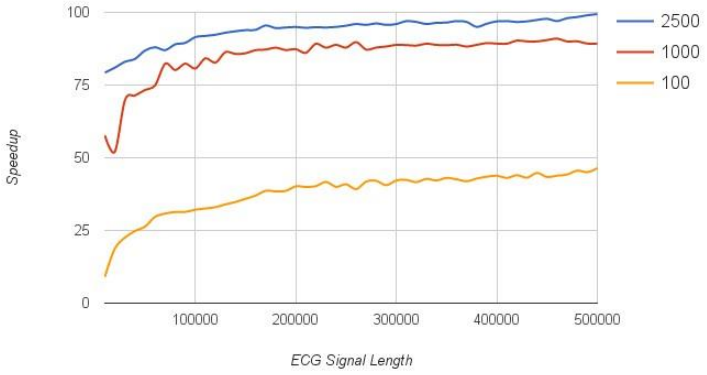


Fig.5. Speedup of parallelized CUDA solution compared to sequential version.

The algorithm has a steady linear speedup as the ECG signal length increases. Additionally, increasing the kernel length has a noticable effect on the speedup. This is due to the effect of allocating and deallocating a thread, which becomes negligible as kernel length increase. We observe that for some configurations a speedup of up to 100 is achieved, with a scalable nature. Therefore the hypothesis is confirmed.

Optimal number of Threads Per Block In the context of our research question we have provided several additional experiments. Selecting right parameters for number of threads in a block has positive effect on the speedup. In order to find the optimal value of the TPB number, we have conducted tests for various TPB from 8 up to 1024. Figure 6 presents the speedup values for different TPB values.

Our analysis shows that, TPB should be kept as high as possible by taking into account the upper limit of threads per block being provided by the specification of the GPU.

7. Conclusion

This work contributes CUDA GPU parallelization for noise filtering of ECG heart signals. The provided parallel solution takes advantage of thousands of CUDA cores. Their size is increasing linearly to the input length used.

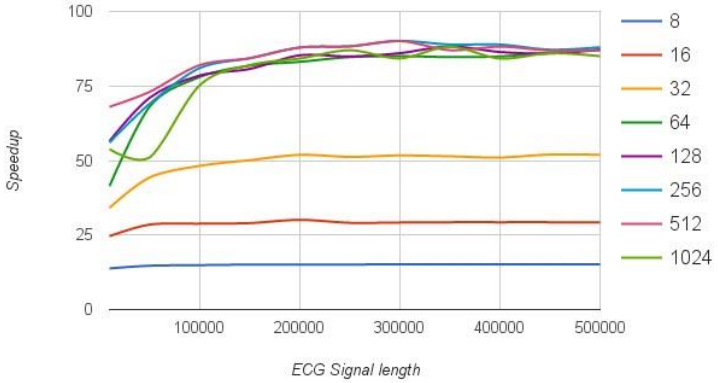


Fig.6. Effect of threads per block on speedup.

Results obtained by executing the sequential algorithm and the parallel CUDA algorithm show they are identical for low-pass, high-pass and band-pass filters.

The analysis on GPU shows higher numbers for threads per block produce higher speedups. Increasing the kernel length has a noticable effect on the speedup. The row version of the CUDA algorithm achieves a speedup proportional to the input and filter length.

This research is the first step of the ECG signal processing with the intention to extract hidden information.

As future work, we plan to carry out more tests on multiple GPUs tied together and compare the results to the OpenMP and similar solutions. We also plan to find a more appropriate CUDA algorithm and optimize the GPU execution if possible by an element algorithm version, where the threads are defined on element level, and not on a row level, expecting that the parallelized code will scale linearly with increasing filter size, and achieve even higher speedups by eliminating the synchronization barriers and aligning the memory access without bank conflicts.

References

1. The cuda zone, https://developer.nvidia.com/cuda-zone, last visited on 25.04.2016
2. Acharya, R., Krishnan, S.M., Spaan, J.A., Suri, J.S.: Advances in cardiac signal processing.Springer (2007)
3. Antoniou, A.: Digital signal processing. McGraw-Hill Toronto, Canada: (2006)
4. Domazet, E., Gushev, M., Ristov, S.: Dataflow dsp filter for ecg signals. In: 13th InternationalConference on Informatics and Information Technologies. Bitola, Macedonia (2016)
5. Duato, J., Pena, A.J., Silla, F., Mayo, R., Quintana-Orti, E.S.: Performance of cuda virtualizedremote gpus in high performance clusters. In: Parallel Processing (ICPP), 2011 International Conference on. pp. 365–374. IEEE (2011)
6. Lugovaya, T.S.: Biometric human identification based on ecg (2005)
7. Phillips, E.H., Fatica, M.: Implementing the himeno benchmark with cuda on gpu clusters. In: Parallel & Distributed Processing (IPDPS), IEEE International Symposium on. IEEE (2010)
8. Smith, S.W.: Digital signal processing: a practical guide for engineers and scientists (2003)