ARTIFICIAL SATELLITES, Vol. 48, No. 2 – 2013
DOI: 10.2478/arsa-2013-0005

GNSS SIGNAL PROCESSING IN GPU

Petr Roule, Ondřej Jakubov, Pavel Kovář, Petr Kačmařík, František Vejražka
Czech Technical University in Prague, Faculty of Electrical Engineering
e-mail: roulepet@fel.cvut.cz

ABSTRACT

Signal processing of the global navigation satellite systems (GNSS) is a computationally demanding task due to the wide bandwidth of the signals and their complicated modulation schemes. The classical GNSS receivers therefore utilize tailored digital signal processors (DSP) not being flexible in nature. Fortunately, the up-to-date parallel processors or graphical processing units (GPUs) dispose sufficient computational power for processing of not only relatively narrow band GPS L1 C/A signal but also the modernized GPS, GLONASS, Galileo and COMPASS signals.

The performance improvement of the modern processors is based on the constantly increasing number of cores. This trend is evident not only from the development of the central processing units (CPUs), but also from the development of GPUs that are nowadays equipped with up to several hundreds of cores optimized for video signals. GPUs include special vector instructions that support implementation of massive parallelism. The new GPUs, named as general-purpose computation on graphics processing units (GPGPU), are able to process both graphic and general data, thus making the GNSS signal processing possible.

Application programming interfaces (APIs) supporting GPU parallel processing have been developed and standardized. The most general one, Open Computing Language (Open CL), is now supported by most of the GPU vendors. Next, Compute Unified Device Architecture (CUDA) language was developed for NVidia graphic cards. The CUDA language features optimized signal processing libraries including efficient implementation of the fast Fourier transform (FFT).

In this paper, we study the applicability of the GPU approach in GNSS signal acquisition. Two common parallel DSP methods, parallel code space search (PCSS) and double-block zero padding (DBZP), have been investigated.

Implementations in the C language for CPU and the CUDA language for GPU are discussed and compared with respect to the acquisition time. It is shown that

the GPU implementation was approximately sixteen times faster than the CPU's for signals with long ranging codes (with 10230 number of chips - Galileo E5, GPS L5 etc.).

Paper presented at the "European Navigation Conference 2012", held in Gdansk, Poland

INTRODUCTION

The aim of our work is to develop a software-defined radio (SDR) receiver of global navigation satellite systems (GNSS) which utilizes the performance of new generation graphic processor units (GPUs). The forecasted massive penetration of these highly parallel digital signal processors (DSP) has motivated us to start implementation of the most computationally demanding signal processing tasks. The final goal is to create a multi-frequency and multi-system receiver which will benefit from all available GNSS systems (GPS, Galileo, GLONASS, COMPASS,…) and will provide higher accuracy estimates of position, velocity and time.

The application of such an approach in GNSS has already appeared in the literature. The GPU-based single-system and single-frequency (GPS L1 C/A) receiver has been implemented (Hobiger et al. 2010). The CUDA programming language enhancement was adopted to NVidia GPU. In this paper, we also implemented the acquisition algorithms using this approach for simplicity, but investigated the acquisition times of other GNSS signals as well.

The high prospective of GPUs as GNSS signal processors is their high performance, relatively simple programming, low price and power consumption, wide availability as GPUs are integrated to PCs, notebooks, PDAs and smart phones. The main suppliers of that technology include AMD, Intel, NVidia and ARM. In the future, this technology will likely be broadly integrated into handheld devices and will autonomously substitute the current third-party DSPs.

The overall receiver architecture would require a suitable front end, analog-to-digital convertors (ADC) and a communication interface to the host GPU. An example might be the USB 3.0 interface with sufficient throughput and available device drivers.

In the first section, we introduce parallel GPU and CPU architectures. In the second section, we overview the most common parallel methods of GNSS signal acquisition that are suitable for parallel implementation. Then, we reveal the crucial implementation issues connected with these methods. Experimental results comparing the number of steps of all the methods and their acquisition times are delivered in the last section.

1. PARALLEL PROCESSING IN GPU AND CPU

The performance improvement of modern processors is based on the increasing number of their cores. For exploitation of that performance it is necessary to write programs with parallel approach. The current PCs can offer two kinds of multi-core processors. - multi-core CPU (Central Processor Unit) and GPU (Graphic Processor Unit). The possibility to employ GPU in general purpose computing (not just for graphic applications) came up a couple of years ago and opened a new way of adopting multi-core performance on graphic cards to other applications. - GPGPU (**General-purpose computing on GPU)**).

The paradigm of effective parallelism of the signal processing tasks comprises both types of the following decompositions:

- Task decomposition: dividing the algorithm into individual tasks (don't focus on data)
- Data decomposition: dividing a data set into discrete chunks that can be operated on in parallel

Various hardware processors CPU or GPU are generally better suited for some types of parallelism more than the others. Contradictory examples are shown in the table below (Mistry et al., 2011).

Table 1. Examples of hardware architectures and their suitability for parallelism

Hardware type	Examples	Parallelism
Multi-core superscalar processors	AMD Phenom II CPU	Task
Multi-core SIMD processors	Radeon 5870 GPU	Data

This crucial difference between the CPU and GPU hardware architecture is depicted in Figure 1. Note that CPU has less number of but more powerful arithmetic logic units (ALUs) and a large cache with control logic, whereas GPU comprises a large number of relatively simple ALUs with little cache and simple control logic, thus making GPUs prospective for parallel signal processing with primitive arithmetic operations such as multiplication, accumulation, discrete correlation and convolution, FFT, etc.

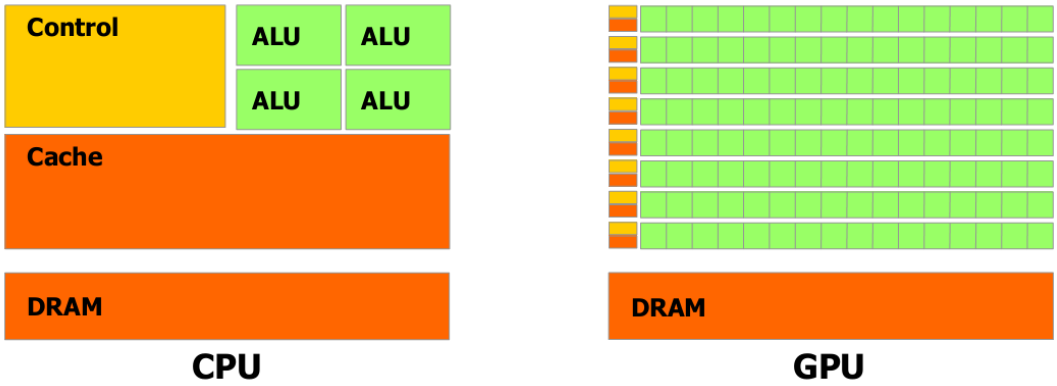

Fig. 1. Comparison of CPU and GPU hardware architectures (Nvidia CUDA, 2011)

A common GPU processor consists of one or more multiprocessors (SMs). Each multiprocessor is designed for execution of hundreds of threads concurrently. To manage such a large number of threads, it employs a unique architecture called SIMD (Single-Instruction, Multiple-Data). Multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads (for Nvidia Geforce 9800) called warps. Individual threads composing a warp start together and register state and are therefore free to branch and execute independently.

The GPU programming is done through an APIs (Application Programming Interfaces) that has already been developed and standardized. The most general one, Open Computing Language (Open CL), is now supported by most GPU vendors. The next one, Compute Unified Device Architecture (CUDA) language was developed solely for NVidia graphic cards. The CUDA language includes highly optimized signal processing libraries including efficient implementation of the fast Fourier transform (FFT).

2. METHODS OF GNSS SIGNAL ACQUISITION

The goal of the GNSS signal acquisition is to estimate the initial PRN code delay m and carrier frequency offset o that are required for initialization of the carrier and code tracking loops. The maximum likelihood (ML) estimate of the acquisition parameters is given:

$$\left(\hat{o},\hat{m}\right) = \arg\max_{o,m}\left|\sum_{n=0}^{N-1} x(k)c^{*}(k-m)e^{-j2\pi of_{\Delta}kT_s}\right| \tag{1}$$

where $x(k)$ is the input signal, $c(k\text{-}m)$ is the PRN code replica generated in the receiver, f_{Δ} describes the frequency step and T_s is the sampling period. The algorithm searches over o and m where the absolute value of the cross-correlation function gives maximum.

The acquisition is the most complicated signal processing task from the computation point of view. For test implementation and performance investigation the two common methods were chosen - PCSS (Parallel Code Space Search) that is algorithmically very simple, DBZP (Double Block Zero Padding), which is based on the calculation of a large number of short FFTs. Both methods are therefore suitable for processing in GPU.

Decomposition of the PCSS Method

The PCSS is a method for effective calculation of the circular cross-correlation function between the input signal and the PRN code replica with a single frequency offset. The circular cross-correlation is evaluated using FFT (Kai et al., 2007). The PCSS method is divided in the five following steps (complying with the numbers in Fig. 2.).

1. Multiplication of the input signal by the signal of the local oscillator (compensation of the frequency shift residual)

2. FFT of the mixed signal

$$X(n) = \sum_{k=0}^{N-1} x(k) e^{-j2\pi kn/N} \tag{2}$$

3. Multiplication of the input signal spectrum by the complex conjugate of the replica spectrum

$$X(n)C^*(n) \tag{3}$$

4. Inverse transform to the time domain

$$R(m) = \frac{1}{N} X(n)C^*(n)e^{j2\pi mn/2} \tag{4}$$

5. Calculation of the absolute value

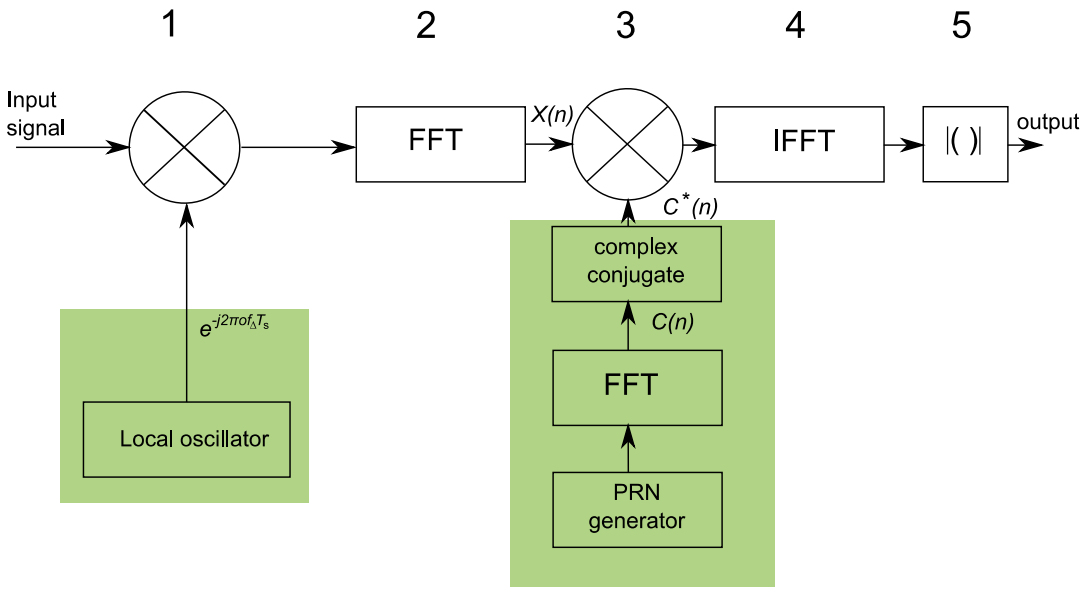

Fig. 2. Acquisition method PCSS

The green blocks of the algorithm in Fig. 2 can be calculated in advance, hence they are not considered in the performance analysis.

Decomposition of the DBZP Method

The DBZP (Double Block Zero Padding) method is depicted in Fig. 3. It is a highly effective method for calculation of the cross-correlation function. The idea of this method is to divide input signal and PRN code to smaller parts from which the algorithms calculates the partial linear cross-correlation functions in the frequency domain. The partial cross-correlation functions are then summed together over the frequency shift. (Chao-jun Wei et al., 2010).

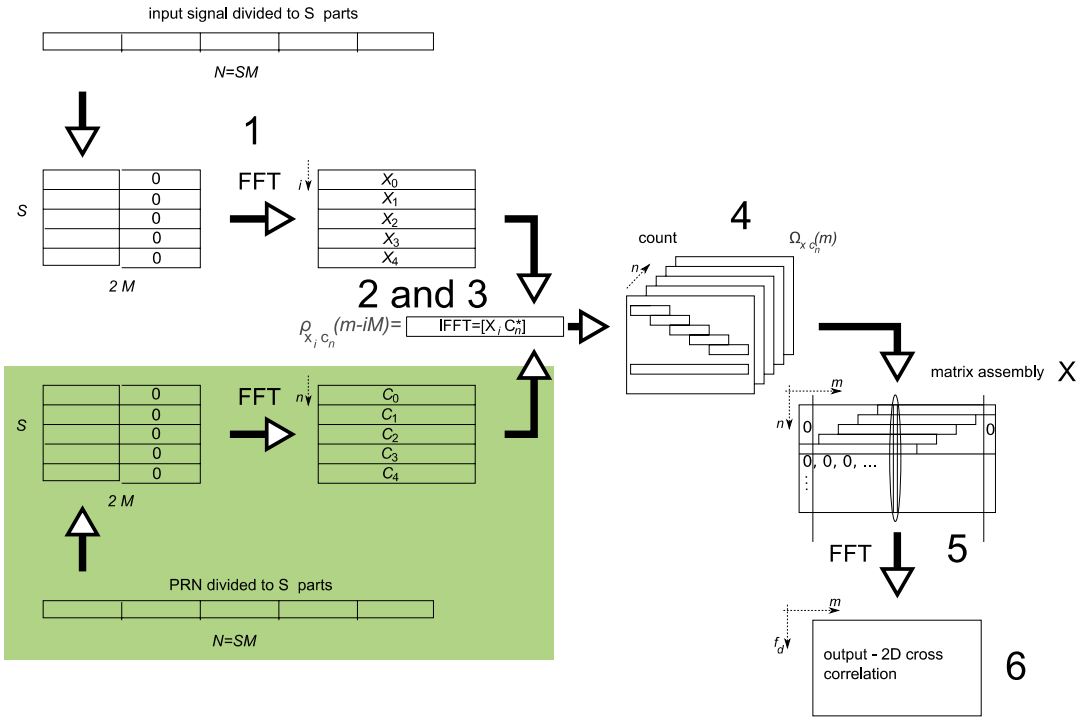

Fig. 3. Acquisition method DBZP

We denote M as the size of a part of the signal, symbol S denotes the number of the parts. The green part in Fig. 3 can be calculated in advance - not in real time.

The DBZP method is algorithmically given by the following steps:

1. Selection of the parts of the input signal and of parts of the PRN code

$$x(k) = \sum_{i=0}^{S-1} x_i(k - iM) \quad and \quad c(n) = \sum_{i=0}^{S-1} c_i(k - iM) \tag{5}$$

And the transform of each individual part filled with zeros of length M to the sprectral domain according to (2) where we substitute $2M$ for N.

2. The multiplication of the spectra of the part of input signal and its complex conjugated replica parts.

3. Switch to the time domain using IFFT, we obtain the cross-correlation between i part of input signal and n part of PRN code

$$\rho_{x_i c_n}(m) = \sum_{k=-\infty}^{\infty} x_i(k) c_n^*(k - m) \tag{6}$$

4. Rearrangement of the results

$$\Omega_{xc_n}(m) = \sum_{i=0}^{S-1} \rho_{x_i c_n}(m - iM) \tag{7}$$

5. Summation of the partial correlation function over the frequency shifts

$$\sum_{n=0}^{S-1} \Omega_{xc_n}(m+nM) e^{-j2\pi o f_\Delta T_s nM} \tag{8}$$

6. Calculation of the absolute value.

3. IMPLEMENTATION ISSUES

This section describes the implementation issues of both methods. In the first approach, we will not consider the processor overhead and memory operations. We consider just the number of independent steps required for either acquisition method. For sequential processing in one core (CPU), the number of independent steps equals the number of FLOPs (FLoating-point OPerations). In both methods, only three types of arithmetic operations are involved - FFT or IFFT, complex multiplication and complex summation.

Concerning the number of steps of the FFT and IFFT, we will consider the radix 2 butterfly algorithm that is featured with the two independent steps (multiplication, sum). The FFT with N samples has $3\log_2 N$ independent steps.

The complex multiplication can be decomposed into two independent steps and the complex sum can be done in one independent step.

The GPU has a finite number of cores. The number of cores in a processor unit is a restrictive parameter. The parallel programming approach (OpenCL, CUDA) is based on the following hierarchy and terminology. A task for one core is called a thread. The threads are associated to groups called blocks. The size of the blocks is optional but has to be less than the number of cores of the processor or equal. The blocks are executed in parallel or consecutively. It depends on the number of cores of the GPU.

In our preliminary study, we used the GPU NVidia Geforce 9800 with 128 cores with maximum block size of 128 threads. The comparison of the number of steps for both acquisition methods is depicted in Fig. 4.

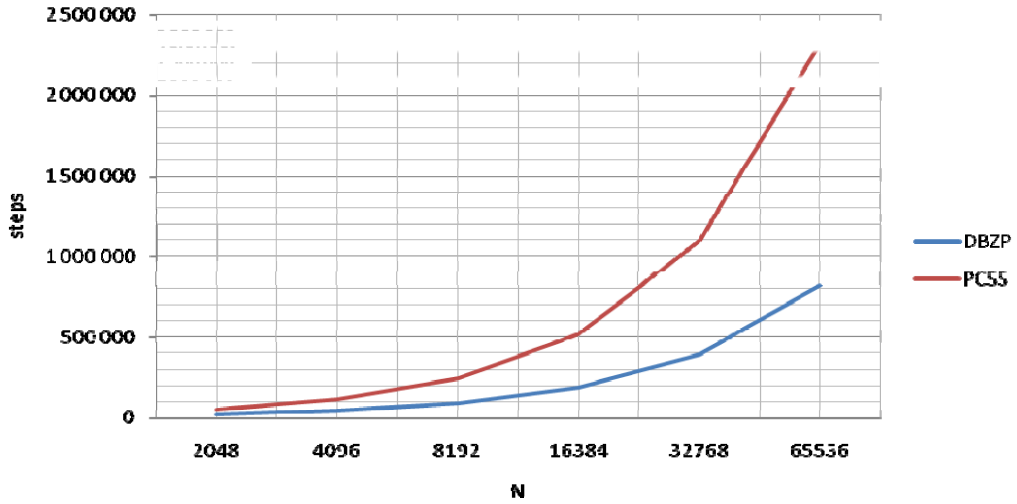

Fig. 4. Comparison of the number of steps for DBZP and PCSS acquisition methods (N is number of processed samples)

Fig. 4 shows that the DBZP method requires fewer steps for one satellite than the PCSS method. But it should be noted that the DBZP method has more complicated algorithms than the PCSS method. It brings higher processor overhead and significantly more memory operations.

4. COMPARISON OF EXPERIMENTAL RESULTS

For testing all the methods we used a PC (Personal Computer) with the following parameters:

Table 2. Description of the PC platform used for the experiment

CPU	Intel core 2 duo 2.0 GHz
Graphic card	Nvidia Geforce 9800
Memory	3 GB

The algorithms were implemented in the C programming language enhanced by the CUDA parallel programming language. The CUDA language option provides the optimized signal processing libraries including efficient implementation of the fast Fourier transform (FFT). These optimized signal processing libraries were the main reason why we chose the CUDA language for our first experiments.

Figure 5 shows a comparison of both CPU and GPU times spent on both acquisition algorithms.

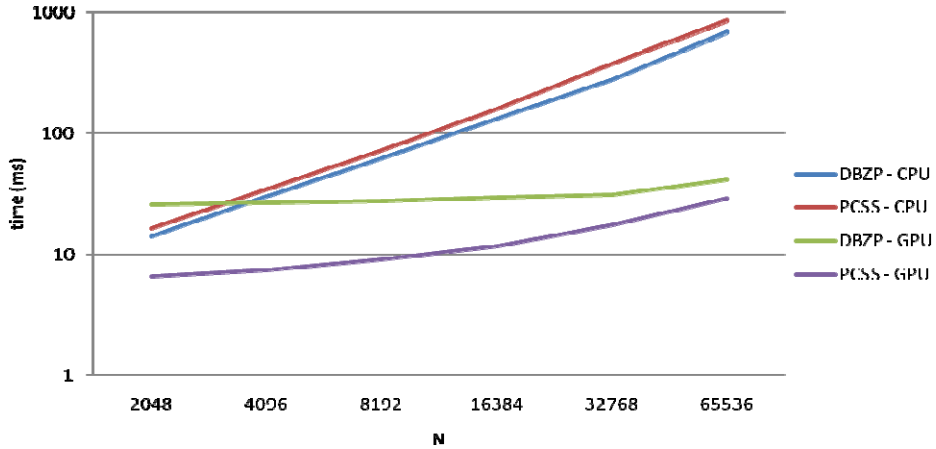

Fig. 5. Comparison of acquisition times for DBZP and PCSS methods in CPU and GPU for 1000 attempts (N is number of processed sample)

In Fig. 5, it is shown that the algorithms in the GPU run much faster than in the CPU, whereas in the CPU the DBZP method is more efficient (average 50 %), in the GPU the PCSS method (average 20 %) outperforms the DBZP method. The reason why the DBZP method is faster than the method PCSS was explained in the previous chapter. The modern GNSS signals commonly use long PRN codes, for instance 10 230 or longer. From Nequist sampling theorem number of samples for code 10 230 chip should be equal to or greater than 20460 sample (this corresponds to sampling rate 2 samples/code). From Fig. 5, it is apparent that the computational time of the GPU is approx. 15 times lower for 32 768 samples (sampling rate 3,2) and even 23 times lower for 65536 samples (sampling rate 6,4) than in the CPU. All the algorithms computed FFT transformation over 2^n samples. FFT computed over any number of samples have been tested as well but results show that computation FFT over 2^n samples is more effective for CUDA_FFT library.

CONCLUSION

Our preliminary study shows that GPU is very suitable to be employed in GNSS signal processing. In addition to it, the performance of the GPU is continually growing, thus in the near future the processing of all the GNSS signals will be possible. This fact allows us to work on the implementation of the multi-frequency and multi-system GNSS SDR receiver based on GPU that will meet the requirements for a real-time operation.

REFERENCES

Hobiger T., Gotoh T., et al. GPU based real-time GPS software receiver,GPS SOLUTIONS (2010), pp. 208-216

Mistry P., Schaa D., et al. OpenCL University Kit, [Cited 20-04-2012], AMD 2011, Available at: http://developer.amd.com/zones/OpenCLZone/universities/Pages/default.aspx

NVidia: CUDA C, Programming Guide, Version 4.1, 2011, Available at: http://www.nvidia.com

Kai B. Dennis M. Akos, et al.A Software-Defined GPS and Galileo Receiver, Boston, Birkhauser 2007

Chao-jun Wei, Song-lin Sun et al. An Enhanced Spectrum Method for GPS Weak Signal Acquisition, in Proceedings of 2nd IITA International Conference on Geoscience and Remote Sensing 2010, Qingdao (China), pp. 502-505

T. Hobiger, et al., A real-time GNSS-R system based on software-defined radio and graphics processing units, Advances in Space Research, Volume 49, Issue 7, 1 April 2012, Pages 1180-1190, 10.1016/j.asr.2012.01.009, 2012.

Received: 2012-07-31,

Reviewed: 2013-01-16, by T. Hobiger,

Accepted: 2013-02-20.