PROBLEMY MECHATRONIKI
UZBROJENIE, LOTNICTWO, INŻYNIERIA BEZPIECZEŃSTWA

ISSN 2081-5891 **10, 1 (35), 2019, 91-98**

PROBLEMS OF MECHATRONICS
ARMAMENT, AVIATION, SAFETY ENGINEERING

Acceleration of Signal Processing Algorithms in Seekers Using Graphics Processing Units

Piotr TUREK

Military University of Technology, Faculty of Mechatronics and Aerospace,
2 gen. Witolda Urbanowicza Str., 00-908 Warsaw, Poland
Corresponding author's e-mail address and ORCID:
piotr.turek@wat.edu.pl; https://orcid.org/0000-0002-2869-3182

Received by the editorial staff on 10 April 2017
The reviewed and verified version was received on 14 February 2019

DOI 10.5604/01.3001.0013.0799

Abstract. The paper presents a discussion on the issue of possible acceleration of radiolocation signal processing algorithms in seekers using graphics processing units. A concept and implementation examples of algorithms performing digital data filtering on general purpose central and graphics processing units are introduced. The results of performance comparison of central and graphics processing units during computing discrete convolution are presented at the end of the paper.

Keywords: graphics processing units, signal processing, Compute Unified Device Architecture

1. INTRODUCTION

The purpose of this paper is to present the possibilities of using accelerating signal processing algorithms using GPUs (*Graphics Processing*

This work has been compiled from the paper presented during the 21th International Workshop on *CAD, CAM and CAE Systems*, Jurata, Poland, May 8-12, 2017.

Unit) and a comparison of times to complete digital signal filtration programs performed on GPUs when compared to their equivalents implemented on CPUs (*Central Processing Unit*).

Digital processing of radiolocation signals involves the transmission of large data streams, which even for signals of intermediate frequencies may amount to several hundred megabytes per second. There are two methods of processing such signals: hardware method, implemented on dedicated structures, e.g. FPGA (*Field Programmable Gate Array*), and software method implemented on DSPs (*Digital Signal Processor*), using general purpose CPUs and GPUs.

Digital radiolocation signal processing algorithms are well suited for implementation in parallel architectures, as their essence is repeated performance of identical multiplication and accumulation operations. One of the latest trends in parallel data processing is the use of graphics processors with an architecture suitable for this task. In recent years, graphics processors have evolved in a direction enabling complete software control over the computations they perform, which enabled using them for other purposes than generating graphics, and resulted in the emergence of so-called GPGPUs (general purpose graphics processing units). A major step was the development by graphics processor market leaders of libraries dedicated for parallel computation, such as OpenGL and CUDA (*Compute Unified Device Architecture*). These libraries contain implementations of mathematical tools that support the commonly used high-level programming languages (C, C++, Python) and can successfully be applied to signals processing. GPU manufacturers declare that parallel execution of operations can accelerate signal processing algorithms several-fold.

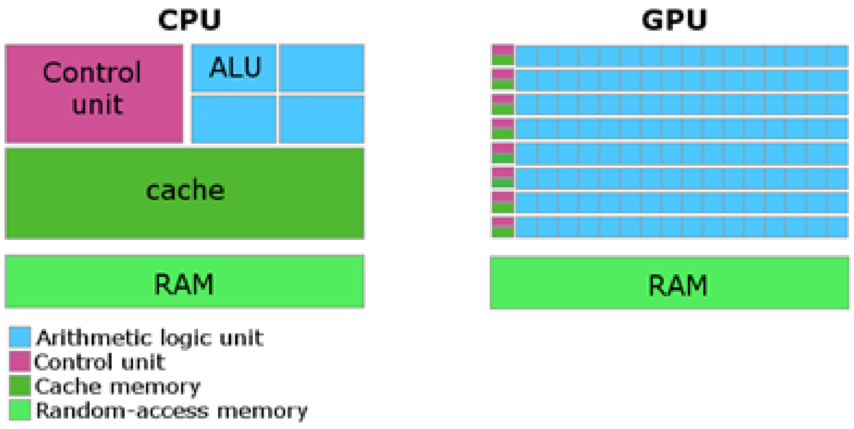

Fig. 1. Differences in CPU and GPU architecture

It must be noted that GPU architecture and capabilities are not equally suited for all tasks. GPUs are dedicated for parallel execution of the same operation on multiple data samples at the same time.

In contrast, CPUs are better suited for performing different operations concurrently. The difference mainly stems from the different architecture of the processors, as shown in Fig. 1. GPUs have much greater numbers of arithmetic logic units (cores), although their clock rates are usually lower than in modern CPUs.

2. IMPLEMENTATION OF FILTERING ALGORITHMS ON CPUS AND GPUS

In radiolocation, the signal received is very frequently subject to interference whose power is greater than the emitted signal echo, which necessitates the use of filtering in the reception track. The paper presents a comparison of the time required by CPUs and GPUs to complete a digital convolution, which is one of the basic operations performed in signal filtering. The convolution function is described by an integer:

$$f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)\,d\tau \tag{1}$$

While a discrete convolution is described by a sum:

$$f[n] * g[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{2}$$

The discrete convolution algorithm was subsequently implemented on the following processors: a quad-core Intel(R) Core™ i5-3230M CPU 2.6 GHz and the NVIDIA GeForce GT 640M LE GPU with 384 cores. To compare the operation completion times, original applications were developed using the C++ and CUDA C languages. 32-bit float samples were processed for discrete signals of different lengths. A fragment of the function performing the discrete convolution on the CPU is shown in Listing 1. It shows that the multiplication and accumulation operations are performed sequentially for subsequent signal samples. The function performance time depends in this case on the processor's clock rate. Performing a discrete convolution on a graphics processor is a more complex operation. The programmer has no direct access to the GPU memory, and the data to be processed must be sent there by the CPU through the RAM. Implementing algorithms on a graphics processor requires having a dedicated compiler installed on the computer, which can differentiate between the code written to be executed on the GPGPU device from the code intended for host CPU.

Both in the OpenCL open standard and in a solution dedicated to devices of a single company only – CUDA, the code syntax does not differ significantly from the classic "C" language.

Functions executed on the GPU are termed "kernels", and their declaration must be preceded and finished by a double "__" sign.

Calling a kernel performing a task on the GPU requires stating at least the number of blocks and threads where the function is to be executed, as shown in Listing 2.

Listing 1

```
float   convolution_cpu(float   *sig_in,   float   *filter,
int sig_length, int filter_length)
   {
       //  . . .
       // Computation of convolution on CPU

       for (i=0; i< sig_length; i++)
       {

           for (j=0; j< filter_length; j++)
           {
               if(i-j >=0)
                   sig_out[i]+= sign_in[j]*filter[i-j];
           }
       }
    }
```

Listing 2

```
Function_name<<< number_of_blocks , number_of_threads>>>.
```

Fig. 2 shows the logical structure of a graphics processor, which is made up of elements contained in one another: grid, blocks and threads.

An additional important issue when writing programs for GPUs is the use of shared and local (private in OpenCL) memory. These two memory types enable much faster read and write operations than global memory. In the application written for the GPU, shared memory was utilised, which is available to all threads in a block. Parallel execution of operations carries the risk that operations executed in individual threads may not be completed exactly at the same time. For this reason, thread synchronisation is implemented, which prevents initiating subsequent phases of the program before threads in the entire block complete their operations.

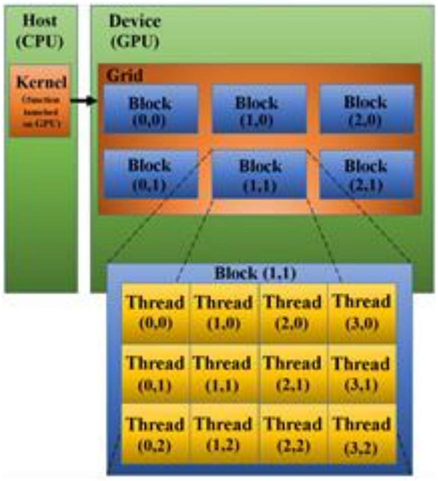

Fig. 2. Organisation of logical structures and processing units in a GPU

Listing 3 shows fragments of code that executes the discrete convolution function on the GPU.

Listing 3

```
__global__ convolution_gpu(*sig_in, *filter, *sig_out)
{
    //  . . .
    //  variable  “index”  allowing  for  reference  to  thread
    //  elements
      index = blockDim.x*blockId.x + threadId.x;

      __shared__ float sig_shared[width];
    // ...
       sig_shared[threadIdx.x]=sig_in[index_in_x];
       __syncthreads();

    //  Computation of convolution on GPU

       if(threadIdx.x<O_Tile_Width)
       {
     sum=0.0f;
       for(int j=0;j<width;j++)
       {
          sum+=M[j]*sig_shared[j+threadIdx.x];
       }
     sig_out[index]=sum;
}
```

A noticeable difference in comparison with the code written for the CPU is the occurrence of only one for loop. Incrementing subsequent samples is not required as they have been distributed to further threads in blocks in the GPU cores. This is enabled by the index variable used in the code, which accesses the indexes of logical blocks and threads in the GPU's grid.

Each time a function with arguments related to the dataset is called, it requires previous allocation of memory at the GPU. These data are copied from the CPU through the RAM. When the program is completed, the allocated memory should be freed.

3. RESULTS OBTAINED

The tests of digital convolution execution on CPUs and GPUs involved measuring the time to perform the convolution operation. A constant number of 400 samples of impulse response of a filter matching a signal coded with a Barker code was used in the test.

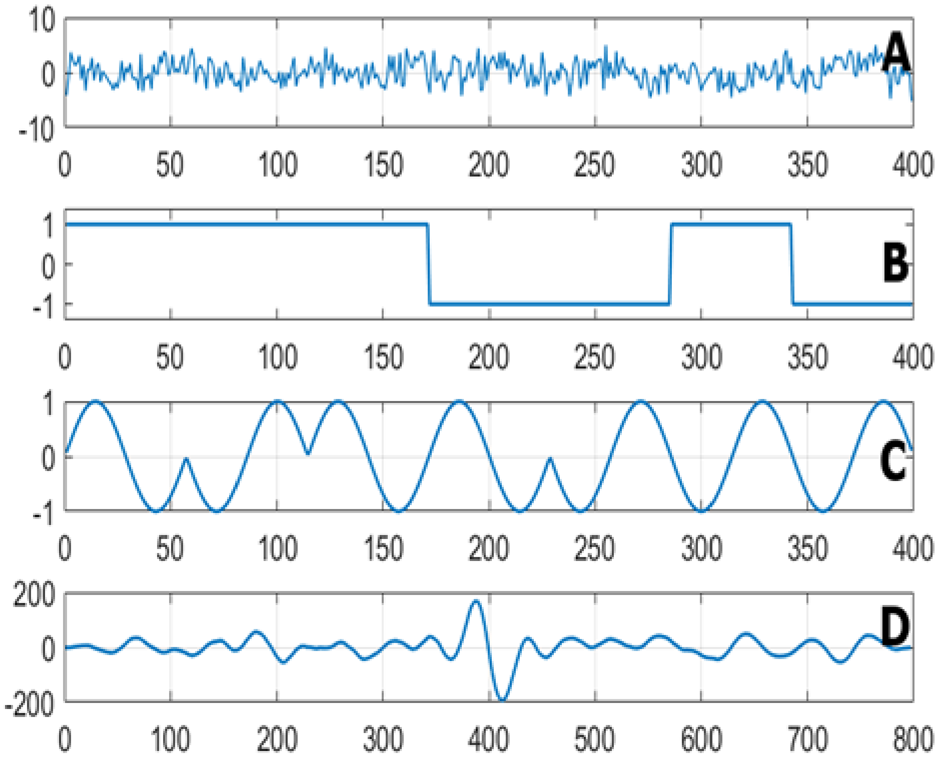

Fig. 3. Matched filtering using a Barker code

Additive noise was added to the signal to reduce the signal-to-noise ratio, which enabled illustrating the capabilities of matched filters in the detection process. The number of convoluted radiolocation signal samples was increased for subsequent measurements.

Figure 3 shows sample results of convoluting 400 samples of noise-burdened radiolocation signal (part A), whose phase was coded with a Barker code (part B) with a reference signal, shown in part C. The resulting signal is shown in part D. As a result of the operation, it is possible to unambiguously determine the period of repetition of the reference signal in the noised signal.

The results of convolution completion time measurements on the GPU and CPU for different numbers of radiolocation signal samples are shown in Fig. 4.

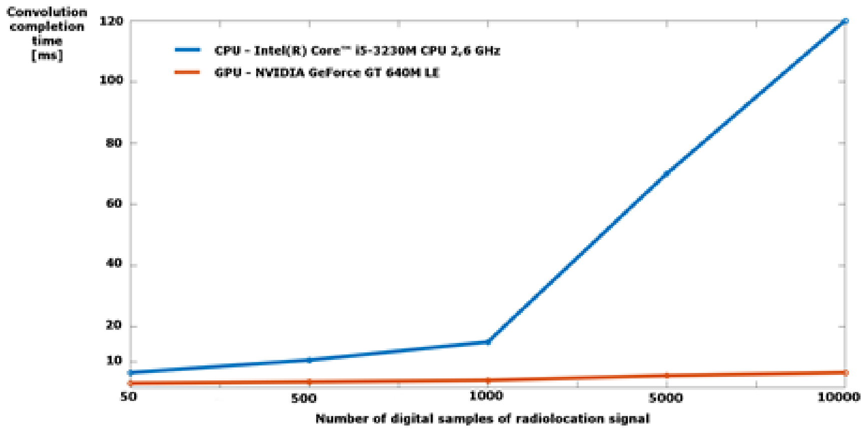

Fig. 4. Discrete convolution operation completion time for a CPU and GPU at different radiolocation signal sample lengths

4. SUMMARY

The paper presented examples of implementing a radiolocation signal filtering algorithm utilising the discrete convolution operation with a matched filter. The results shown confirm that it is possible to accelerate radiolocation signal processing by using GPGPUs instead of CPUs. The level of acceleration depends on numerous factors, such as the ratio of the number of samples to the number of threads in the GPU blocks, the GPU architecture, and the degree to which local and shared memory is used. However, increased acceleration with the number of convoluted samples is noticeable as well. Such a result confirms the viability of using GPUs for processing digital radiolocation signals, where the number of signal samples in a unit of time is very high. The results obtained confirm a greater time-efficiency of the discrete convolution operation when performed on a graphics processor, as compared to implementing this operation on a CPU.

REFERENCES

[1] Jason Sanders, Edward Kandrot. 2011. *CUDA w przykładach: Wprowadzenie do ogólnego programowania GPU*. NVIDIA Corporation.

[2] http://docs.nvidia.com/cuda/ - CUDA Toolkit Documentation v8.0.61 [accessed 28.03.2017] ,

[3] Guillon A.J. 2015. *An Introduction to OpenCL C++*, The Khronos Group Inc.

Akceleracja algorytmów przetwarzania sygnałów w głowicach samonaprowadzania z wykorzystaniem procesorów graficznych

PIOTR TUREK

Wojskowa Akademia Techniczna, Wydział Mechatroniki i Lotnictwa
ul. gen. Witolda Urbanowicza 2, 00-908 Warszawa 46

Streszczenie. W artykule zamieszczono rozważania na temat możliwości akceleracji algorytmów przetwarzania sygnałów radiolokacyjnych w głowicach samonaprowadzania z wykorzystaniem procesorów graficznych. Przedstawiono koncepcję oraz przykłady implementacji algorytmów realizujących cyfrową filtrację na procesorach klasycznych oraz graficznych ogólnego przeznaczenia. Wyniki porównania wydajności centralnych i graficznych jednostek przetwarzania podczas obliczania dyskretnego splotu przedstawiono na końcu artykułu.

Słowa kluczowe: procesory graficzne, przetwarzanie sygnałów, Compute Unified Device Architecture