FOLIA 180

Annales Universitatis Paedagogicae Cracoviensis

Studia Technica VIII (2015)

Karol Grondzak, *Eduard Vesel*

Scientific Calculations on a Graphic Processor Unit

Introduction

Recent world is highly concurrent. The consumer market requires innovations and improvements of the products. They are developed in the research laboratories by scientists. To develop new product and market it as first the fast infrastructure for scientific calculations is needed. This trend can be observed on the top 500 list of fastest computers ([1]). Scientist from diverse areas of research, including weather forecasting, cancer treatment, DNA analysis, material sciences and many others are utilizing the fastest computers to perform their calculations and simulations in reasonable time.

From the beginning of the computer era, new generations of computers were designed profiting from the Moore's law ([2]) of increase in the density of the elements on the chip. Unfortunately there are some physical limits of this approach and we have almost reached them. New generations of computers must utilize some other principle. One of the possibilities is to go parallel. The increased number of elements on single chip allows for the construction of multi-core processor. The idea is to provide several processing units in one processor and perform calculations in parallel. This is quite new paradigm and requires new approach to the design of algorithms. In the next paragraphs we will introduce one of the emerging technologies – massively parallel architecture of a processor, consisting of hundreds of simple processors executing the same instruction on different data.

Modern Processing Units Architecture

In modern computer there is usually more than one processing unit. The designers of Central Processing Units (CPU) have offset the workload of CPUs to specialized processors, like Direct Memory Access (DMA) or Graphic Processing Unit (GPU). The role of DMA is to transfer data between peripheral devices and main memory on behalf of the CPU. To produce output on the computer monitor the GPU is utilized. Producers of modern GPUs faced the problem with the performance of their devices before the CPU producers. Their solution is the new architecture of

GPUs based on massively parallel paradigm. The role of GPU is to generate output on the screen. This task is the same for all the pixels on the screen, but for each pixel, different data are processed. Thus massively parallel architecture is suitable for this kind of calculations.

To understand the architecture of modern GPUs, we will briefly introduce the concept of parallel processor architectures. Processor is a device which is reading the stream of instructions and performs it on the stream of data. Any of the streams can be either serial or parallel. Having considered this we have four different architectures ([3]):

1. SISD, single instruction stream is executed on single data stream. This architecture is actually not parallel, but it is mentioned here just to be consistent. Typical examples of such architecture are processors of Intel family before the era of multi-core processors. All the processors with many sophisticated technologies also fall into this category, like pipeline instruction execution units, etc.
2. SIMD, single instruction stream processes multiple data. This architecture is common in Digital Signal Processors (DSP) and other devices, where signals are processed. Multiprocessors and multi-core processors are another example of such architecture. First supercomputers, known as vector computers were based on this architecture. Modern GPUs are also designed as SIMD processors.
3. MISD, multiple instructions are executed on the same data. This architecture is mostly used for fault tolerance calculations. Heterogeneous computers process the same data and must agree on the result.
4. MIMD, multiple instructions are executed on different data. This is the most flexible architecture, which enables to process different data by different algorithms. Cluster of computers, grid and cloud are the examples of such architecture. Many of the computers in the top 500 list utilize this architecture.

Soon after the release of new generation of GPUs with massively parallel architecture, their potential for scientific calculations has been recognized. First successful implementation of parallel algorithm was the matrix multiplication [4]. The era of General Purpose GPU (GP-GPU) calculations started. As it is with every emerging technology, the first steps were tedious. There were no specialized tools to prepare programs for GPUs so the programs were prepared in the language describing the primitive graphical operations.

Fortunately, the potential of the expansion of the market was recognized by the GPU producers and new API and development tools were released. As an example we can mention the Compute Unified Device Architecture (CUDA) execution model introduced by Nvidia Company for their products [5].

The basic difference between modern CPU and modern GPU can be seen in Figure 1. The CPU consists of one Control unit, several Arithmetic Logic Units (ALU) and cache memory. GPU contains hundreds of ALUs organized into several groups. Each group shares the control unit and local memory.

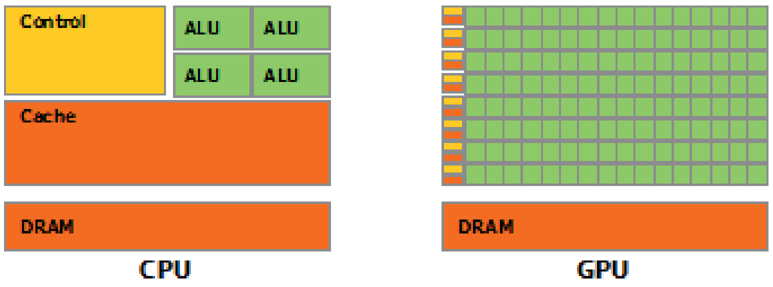

Fig. 1. CPU versus GPU architecture

Source: Nvidia Company

Massively Parallel Calculations on GPU

To demonstrate the possibility of scientific calculation speed-up we will present the step by step process of designing massively parallel code for vector distance calculation. The problem can be formulated as follows. Let us consider two sets of vectors of the same length. We want to calculate the Euclid distance of the vectors in the first set from the vectors in the second set.

The task is accomplished by simple algorithm which, in its serial version, consists of three nested loops (Fig. 2), where the vectors are stored in matrices *First* and *Second* containing N and M rows respectively, and sqr is the operation of the second power. Vectors have K elements. Resulting matrix is of N rows and M columns dimension. The overall amount of operations needed to perform the calculation is then:

$$O_a = 2NMK, \tag{1}$$

$$O_m = NMK, \tag{2}$$

where O_a is the amount of addition/subtraction operations and O_m is the amount of multiplication operations. On SISD architecture the processor would perform all the calculations, iterating the loops. On SIMD architecture we can employ all the available processors and split the outermost loop into several parts, each executed on a separate processor. Having p processors, we can split the loop into N/p parts and we would expect the speed-up of factor p. Because of the many factors, such a speed-up is rarely achieved in the reality (consider the overhead caused by the management of the parallel calculation, the conflicts when accessing main memory, etc.). Implementing the algorithm on SIMD processor this naïve way, we may be dissatisfied by the poor performance. To demonstrate that the detailed understanding of the SIMD processor architecture can be crucial for algorithm implementation, we will compare three implementations of the vector distance calculation algorithm.

```
for (i = 0; i < N; i++)
 for (j = 0; j < M; j++)
  for (k = 0; k < K; k++)
   Result[i, j] = Result[i, j] + sqr(First[i, k] – Second[j, k])
```

Fig. 2. Algorithm for vector distance calculation
Source: author

First implementation is the naïve implementation, which just distributes the workload of two outermost loops to the available processors. It means that each thread calculates the distance of one vector from fist matrix from a vector from the second matrix (Fig. 3).

```
__global__ void distance (int (* first)[N], int (* second)[N], int (* result)[K]) {
    int i, tmp;
    int res = 0;
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    for (i = 0; i < N; ++i) {
                tmp = (first[row][i] – second[col][i]);
                res += tmp * tmp;
 }
 result[row][col] = res;
}
```

Fig. 3. Source code for vector distance calculation on GPU, naïve implementation
Source: author

Studying the memory model of modern GPUs ([6]), we would soon discover that the access to main memory is slow, comparing to access to shared memory, assigned to each group of threads (Fig. 4).

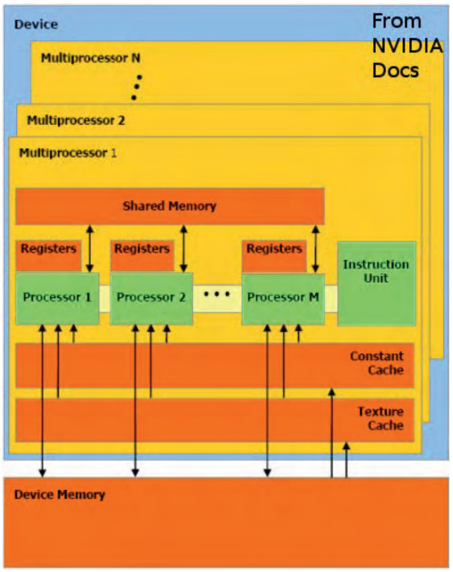

Fig. 4. Memory Model of GPU Streaming Multiprocessor
Source: Nvidia Company

If we consider this model and the organization of the calculation, we can achieve significant improvement of the performance by transferring data needed for calculation into shared memory. The code will change as it can be seen in Figure 5. This minor change drastically improves the performance of the calculation, as can be seen in Table 1.

```
__global__ void distance (int (* first)[N], int (* second)[N], int (* result)[K]) {
 int m,
 int i, tmp, res = 0;
 int row = blockIdx.y * blockDim.y + threadIdx.y;
 int col = blockIdx.x * blockDim.x + threadIdx.y;
 __shared__ int sf[BLOCK_SIZE][BLOCK_SIZE];
 __shared__ int ss[BLOCK_SIZE][BLOCK_SIZE];
 for(m = 0; m < N / BLOCK_SIZE; m++) {
        sf[threadIdx.y][threadIdx.x] = first[row][m * BLOCK_SIZE + threadIdx.x];
        ss[threadIdx.y][threadIdx.x] = second[col][m * BLOCK_SIZE + threadIdx.x];
           __syncthreads();
         for (i = 0; i < BLOCK_SIZE; ++i) {
               tmp = (sf[threadIdx.y][i] – ss[threadIdx.x][i]);
               res += tmp * tmp;
         }
       __syncthreads();
 }
result[row][col] = res;
}
```

Fig. 5. Modified source code for vector distance calculation on GPU using shared memory

Source: author

Tab. 1. The execution times of vector distance calculations for different processors (CPU, GPU) and different implementations on GPU (GPU_1 – naïve, GPU_2 – using shared memory, GPU_3 – using shared memory and minimizing the bank conflicts)

Size of vector and matrices	**Execution time [ms]**			
	CPU	**GPU_1**	**GPU_2**	**GPU_3**
128	7.6339	2.5	0.25	0.12
256	48.594	29.47	1.27	0.51
512	324.39	350	9.15	4.3
1024	2551.6	1245	70	27
2048	21096	6950	563	206
4096	168920	53557	4505	1647

The last, third implementation is targeting the bank conflicts. Shared memory is organized in blocks called banks. Size of each bank is 16. The only access into one bank from different threads which does not cause conflict is when each thread is accessing different cell in bank, or when all threads are accessing the same cell.

The algorithm depicted in Figure 5 does not satisfy this condition and thus there are bank conflicts and the code does not achieve the maximum speed possible. To achieve the optimal bank access, one has to spread data from local arrays located in shared memory into different banks, as it is depicted in Figure 6.

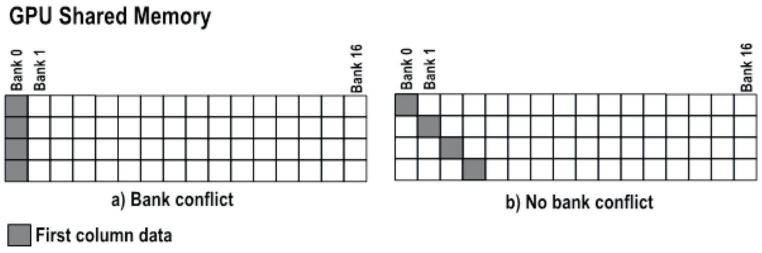

Fig. 6. The bank conflict explanation

Source: author

All three implementations of the vector calculation algorithm were executed and the resulting times are summarized in Table 1. It can be seen that the best results, comparing to the algorithm implementation on CPU, were obtained by third GPU implementation. This result demonstrates that it is possible to achieve significant speed-up of the GPU, when considering the properties and architecture details. This paper is a demonstration of such approach.

Conclusion

In this paper we have introduced the massively parallel architecture of modern GPUs. The SIMD architecture of the modern GPUs can be utilized for parallel scientific calculations. The computational model, development tools and high-level libraries were presented. A simple example demonstrating the possibility of calculations speed-up was presented. Author hopes that this contribution will motivate more scientists to utilize the power of modern GPUs for their research activities.

References

[1] Top 500. The list. Retrieved from http://www.top500.org/.

[2] Brock D.C. (ed.), *Understanding Moore's Law: four decades of innovation*, Chemical Heritage Press, Philadelphia 2006.

[3] Flynn M.J., *Some computer organizations and their effectiveness*, IEEE Transactions on Computers. 1972, C–21(9), p. 948–960.

[4] Larsen E.S., McAllister D., *Fast matrix multiplies using graphics hardware*, Proceedings of Supercomputing 2001, Denver 2001.

[5] CUDA technology. Retrieved from http://www.nvidia.com/object/cuda_home_new.html.

[6] Kirk B.D., Hwu W.W., *Programming Massively Parallel Processors, A hands-on Approach*, Morgan Kaufmann Publishers, 2010.

Abstract

The demands on speed of scientific calculations are growing from the time when the first computer was constructed. Scientists are solving problems which were not viable ten or fifteen years ago. We can also expect that scientists in the future will solve problems which are not viable today. The times of extensive improvement of processors are almost over; we are hitting the physical limits of the electronic devices. Recent trend in improving the computational power of modern processors is to provide several processing units on the same chip, thus parallel thinking when proposing new algorithms is the necessity. In this paper we will introduce the concept of massively parallel Graphic Processing Unit, which can be utilized for particular types of scientific calculations.

Key words: Graphic Processor Unit, Massively Parallel Calculations, Scientific Calculations, CUDA Technology

Karol Grondzak
Pedagogical University of Cracow
Institute of Technology
ul. Podchorążych 2
30-084 Kraków, Poland

Eduard Vesel
Faculty of Management Science and Informatics
University of Zilina
Zilina, Slovak Republic