ATLANTIS PRESS

Advances in Engineering Research, volume 188

International Conference on Aviamechanical Engineering and Transport (AviaENT 2019)

Data Transfer Problems at the Host-Computer Level and Methods to Improve the Performance of Computations on Heterogeneous Systems

Bashkirov Alexey Viktorovich
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

Glotov Vadim Valerievich
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

Astakhov Nikolay Vladimirovich
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

Pirogov Aleksandr Aleksandrovich
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

Kalyadina Anastasia Mihailovny
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

Glotova Tatyana Sergeevna
Faculty of Radio Engineering and Electronics
Voronezh State Technical University
Voronezh, Russian Federation

***Abstract*— In theory, graphics processing units largely exceed central processing units by the degree of performance. But in order to load the GPU with computing tasks to the boundary levels of its maximum performance, the task has to be divided into streams. The number of streams should be comparable to or, to some extent, exceed the value of stream processors of the GPU. Nowadays, modern graphics cards have dozens of thousands of streaming processors. This means that the task of ensuring the division into streams in computation of a sufficient degree of the GPU utilization can be either impossible or very difficult, and the solution to this problem is a "weak point" of the whole method of using the GPU, and offsets its advantages in comparison with the CPU. This scientific article is devoted to this problem.**

Keywords*— processor, video card, pseudorandom number generator

I. INTRODUCTION

Owing to its rapid development, increase in performance and accuracy of the results, the direction of computing using the GPU is now actively mastered by developers and manufacturers of graphics cards that help users by providing access to their internal resources, software environments for creating and writing computer programs to carry out high-performance computations on the GPU. The leaders in the field of target software, creating advanced GPGPU technologies, are AMD and Nvidia, the products of which are OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture). The CUDA technology by Nvidia takes a leading position in terms of performance and stability of computing, but due to the use of only Nvidia architectural solutions by this software and hardware platform, the CUDA is quite limited, and as a result, it is not hardware independent. Thus, this is a serious reason to reject the application and use of this technology in favor of its competitor OpenCL by AMD. This is a fully open, unified implementation of modern GPGPU technology, which allows developers to create cross-platform (platform-independent) software that runs in complex multiprocessor heterogeneous systems.

In theory, GPUs are largely superior to CPUs in terms of performance, but in order to load the GPU with computing tasks to the boundary levels of its maximum performance, the task has to be divided into streams. The number of streams should be comparable to or, to some extent, exceed the value of stream processors of the GPU. Currently, modern graphics cards have dozens of thousands of streaming processors. This means that the task of ensuring the division into streams when computing a sufficient degree of the GPU utilization can be either impossible or very difficult, and the solution to this problem is a "weak point" of the whole method of using the GPU and negates its advantages in comparison with the CPU. System performance will be limited. Usually, when solving a multistream problem, there is a need to exchange digital data with the main calculator of a heterogeneous system. It is necessary to consider the generally accepted architecture of heterogeneous computing system built to work in the

previously selected OpenCL development environment more thoroughly.

This architecture necessarily involves the presence of one to several OpenCL-devices that are connected to the host device. The host device provides services that allow getting access to a variety of OpenCL-devices, which, in turn, are divided into several equivalent computing units with a given large number of processor elements. Calculations on all device hosts are performed by these processor elements of the GPU. OpenCL app commands are sent via host for execution on the processor elements in the device. Each processor element executes only one stream, like SIMD (Single instruction, multiple data) and SPMD (Single program, multiple data).

The main obstacle that complicates the practical use of GPUs for complex computing tasks is a number of limitations associated with insufficient bandwidth of the bus connecting the host and the computing device. For a graphics card, this is a PCI bus (Peripheral component interconnect). For example, for the graphics card used in empirical studies in this work in modeling processes, the PCI Express 2.1 bus has limitations of the maximum bandwidth of each individual line in 5 Gbit/s [1, 2].

An equally important problem that significantly affects the performance of computations using the GPU is the problem of random access to memory and a high degree of global memory latency. To study this issue, it is necessary to consider the memory model provided by the OpenCL standard [2-4].

The OpenCL standard provides the following types of RAM:

1) global:
• write and read;
• extremely low access speed;
2) constant:
• access from all involved processor elements is provided, only read is possible;
3) local:
• write and read for all processor elements of only one group (or computing unit);
• typical high speed of access to this type of memory;
4) private:
• access is allocated by the compiler;
• write and read only within the processor elements;
• high access speed is provided.

Based on the above, the following main problems can be identified:

• access to global memory is extremely difficult because it is limited by high latency and low speed;
• the problem of random access to all types of memory – delay that occurs when multiple processor elements simultaneously access the same memory location (collision);
• problems associated with the task of synchronizing access within a single computing device under conditions of limited access to the memory of a parallel computing unit.

It follows that, when getting access to memory, the main principles are:

• minimizing the number of iterations of access to global memory in the first place as to slower one, and in the second place to local memory;
• application of mandatory data caching – writing the most frequently used blocks of information from the global memory to the local one. In order to consider the data transfer problem at host-computer levels more thoroughly, it is necessary to analyze a more general diagram of the computations performed in the simulation, which must be carried out for one simulation of the decoding process according to the previously selected BP algorithm and its more complete modifications (figure 1). A similar implementation can be seen in the study [1].

In this diagram, blocks 2 – 5 can be divided into streams and run as follows: 2 in *N*- streams, 3 in *M*- streams, 4 in *N*-streams, 5 in *N*- streams. Blocks can run separately due to the absence of group synchronization mechanisms for globalization of data processing streams provided by the OpenCL 1.x. standard. In block 6, the initial and received from the OpenCL device digital data vectors are compared, as well as the decoding errors are calculated and recorded. This procedure occurs consistently, therefore it is more logical to perform in the host part.

The most problematic block in this diagram is block 1, which is used to generate noise. The main element of this block is the pseudorandom number generator. The process of generating a pseudorandom sequence involves the dependence of the register state value on the value of the subsequent generated pseudorandom number and starts the production of sequential generation.

The first block in the diagram should be transferred to the level of the graphics card computing device. Meanwhile, an attempt to generate the necessary pseudorandom numbers using sequential algorithms in one given stream may in practice be worse than the generation of pseudorandom sequences by the host part, and the following procedure for transferring the previously generated pseudo-random sequence from the host to the graphics processor. This situation is primarily connected with a significant loss in terms of the frequency of the stream processors of the GPU compared to CPU processors. The solution to this problem would be the use of a pseudorandom number generator that can simultaneously produce pseudorandom numbers.

Nowadays, this problem is very relevant, and it is actively studied by Russian and foreign scientists [2, 3, 5]. Let us briefly consider the currently existing methods of generating parallel streams of pseudorandom numbers.

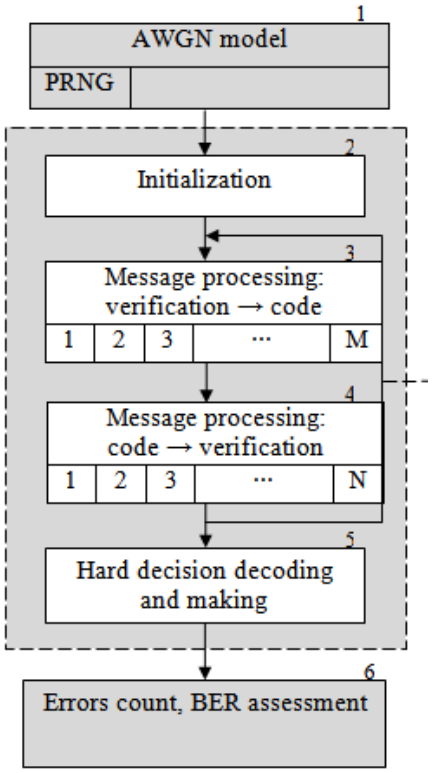

Fig. 1. Diagram of tasks which constitute one simulation while decoding

II. MATERIALS AND METHODS

As [3] shows, a sequential pseudorandom number generator can be used to generate parallel streams of pseudorandom numbers. In applications of parallel methods of digital statistical simulation – Monte Carlo – there is a number of additional requirements, which the ideal generator, applicable for use in pseudorandom number generators, must satisfy, namely:

• scalability of generation of all streams, which, in turn, is necessary to generate the required and sufficient number of streams;

• narrow localization of the generated streams, as a result of which pseudorandom number sequences are obtained without the need of interaction with parallel streams;

• independence of generated sequences created by different streams relative to each other.

Here are the main methods of generating parallel streams of pseudorandom numbers, in accordance with [4, 5]:

A. Randomseeding

Each parallel stream uses the same pseudorandom number generator, with different initialized values selected for each of them.

This method is widely used if there are no strict requirements for the correlation process of streams.

B. Parameterization

A pseudorandom number generator of the same type is used by each stream during generation, and each stream is provided with selecting different sets of parameters. Let us consider an example of the foregoing: it is possible to use different, independent values of the increments c or the multipliers a to operate the linear congruential method.

This method, however, may not provide a lack of correlation between the streams, as well as a given number of generated streams is programmatically limited by a set of parameters, with the ability to configure and change them, that is, in this case, the scalability required for our study is not provided.

C. Block splitting

The sequence of vector data at the output of the r-generator is divided into blocks of a given length M (figure 2) , where M is the largest number of iterations required to solve the problem (the number of implementations of pseudo-random number generators).

The efficiency of this method depends primarily on whether there is a correlation between the elements of the sequence at the distance M.

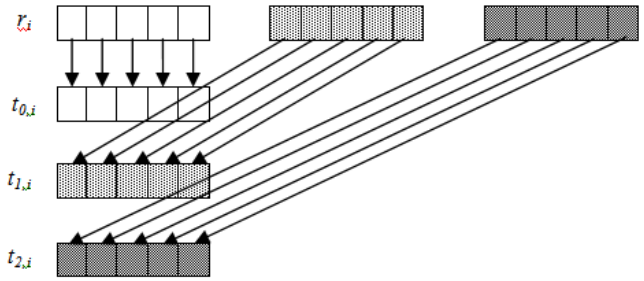

Fig. 2. Parallelization by block splitting

D. Leapfrog

In this method, one generator is used simultaneously by all streams. In this case, each of multiple streams skips p–1 implementations of the pseudorandom number generator, where p is the necessary number of streams (figure 3) [5].

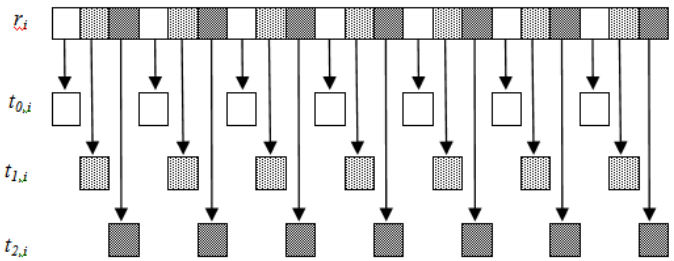

Fig. 3. Parallelization by leapfrog method

The method is highly reliable, and there is no need to estimate the number of implementations of the pseudorandom number generator required to perform the task before computations. But it is also necessary to have an algorithm for

skipping the pseudorandom generator of the previously specified number of values [1, 4, 5, 6].

III. Results and Discussion

The influence on the result of simulating the correlation of streams of generated pseudorandom numbers in the process of modeling noise-resistant low-density codec is not so great, unlike, for example, applications used in cryptography tasks. Therefore, random seeding and parameterization were chosen as a method of parallel separation of the pseudorandom number generator [8,9,10].

Considering the expression:

$$X_n = (ax_{n-r} + c_{n-1}) \bmod b, c_n = \left(\frac{aX_{n-r} + c_{n-1}}{b} \right), \mathrm{n} \geq \mathrm{r}$$

and the algorithm corresponding to this expression:

1. Consider the state of the previous simulation as state S.
2. Parameterization is performed by the initialization block:

$$X_1 = i^3;\ c_1 = i^3.$$

3. 1+*i*mod(3) iterations of the algorithm (1) occur.
4. Generation is carried out in *N/Div* independent streams, and a vector of *N* noise value implementations is created by blocks of *Div* values.
5. The final value of the pseudorandom number obtained in the last iteration stream is automatically entered in the status register. Subsequently, the transformations of the generated pseudorandom numbers are performed according to the Box-Muller method[7].

IV. Conclusion

In order to reduce the correlation of streams, we divide the vector of generated noise into separate blocks. This method leads to a decrease in system performance. Empirically, it was identified that adequate simulation results can be achieved already at *Div* = 2. BER graphs for the presented implementation of the pseudo-random number generator compared to its sequential implementation (figures 4 – 5) prove a sufficient degree of adequacy and practical applicability of the results obtained in the simulation, with much higher performance and speed of their receipt.

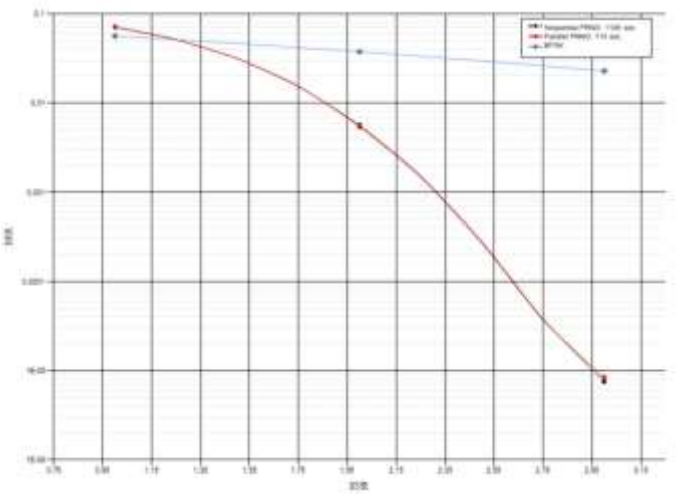

Fig. 4. BER-SNR dependency for code *N* = 1008x504

Decoding in the experiment was performed in 10 iterations according to the BP algorithm, for figure 4: from 1.0 to 3.0 dB with a division of 0.25 dB; for figure 5: from 0 to 2.5 dB with a division of 0.25 dB. In order to ensure the statistical stability of the simulation results, in accordance with the expression 1, variation was carried out at a given level, and the deviation of each bit error rate of 10^{-5} was not more than 10% with an error rate of 0.9.

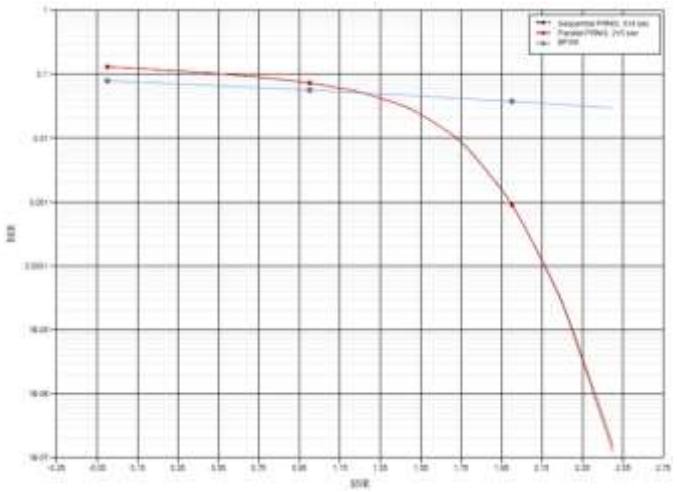

Fig. 5. BER-SNR dependency for code *N* = 8000x4000

In the first case, the level of bit error rate of 10^{-5} was 0.02 dB, in the second case – 0.012 dB, which represented 0.007% and 0.006% respectively and was within the existing framework of a given statistical error. Thus, it is possible to make a clear conclusion that the increased correlation between the bitstreams is not critical in the studied design of the parallel module in the simulation of noise generation. Time saving in the computation is 35% in the first case and 32% in the second case, which is a significant result.

References

[1] S. Kang, J. Moon, In Proceedings of the IEEE International Conference on Communications (ICC) “Parallel LDPC Decoder Implementation on GPU based on Unbalanced Memory Coalescing”, (2012).

[2] A.V. Baschkirov, V.I. Borisov, K.N. Lapshina, A.V. Muratov, V.M. Pitolin. International Journal of Applied Engineering Research, “Influence of Noise Generator Characteristics on the Adequacy of Modelling Noise-Eliminating Codecs with Low Density of Parity Check”, pp. 9622-9629 (2016).

[3] L.Yu, L. Barash, N. Schur, Software Engineering, “On the generation of parallel streams of pseudo-random numbers”, pp. 24-32 (2013)

[4] H. Bauke, S. Mertens, Physical Review E, Random Numbers for Large Scale Distributed Monte Carlo Simulations (2007)

[5] H. Bauke, Tina’s Random Number Generator Library (2014)

[6] L. Barnault, D. Declercq, Inform. Theory Workshop, Fast Decoding Algorithm for LDPC over GF(2q), pp. 70-73 (2003)

[7] G. Eason, B. Noble, and I.N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955. (*references*)

[8] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68-73.

[9] I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[10] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].