Comparative analysis of the implementation of parallel algorithms on the central processors of automation systems in agriculture

Bakhtiyar S. Rakhimov[1*], *Sabokhat* K. Sobirova[1], *Feroza* B. Rakhimova[1], *Asal* A. Allayarova[1], *Atabek* B. Saidov[2], and *Zarina* B. Saidova[2]

[1] Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan
[2]Urgench branch of Tashkent University of Information Technologies named after Muhammad al Khwarizmi, Uzbekistan

Abstract. The article presents a comparative analysis of the implementation of parallel algorithms on the central processors of automation systems in agriculture. Modern automation systems impose increased requirements on the reliability of the implementation of parallel algorithms in real time. It is proposed to use models for the development, analysis and comparison of parallel algorithms on GPUs. The proposed model of parallel computing on GPUs is designed to simplify the development of parallel algorithms for a heterogeneous CPU-GPU environment. Those, with this model, you can: develop parallel algorithms that use data parallelism, while applying the existing experience in creating parallel algorithms for the PRAM machine; evaluate the running time of the parallel algorithm and analyze which part of it is the most resource-intensive and requires optimization; compare parallel algorithms according to some parameters of the model, and, if required, select the best one according to these parameters in real time for automation in agriculture.

1 Introduction

Most of the modes in automated systems that are used in agricultural production proceed in real time [1-4]. Chemical industries associated with the production of mineral fertilizers and concentrates for field cultivation are hazardous industries from the point of view of environmental protection [5, 6]. Automation systems for such production facilities require increased reliability of the hardware and fault-tolerant execution of algorithms and software that supports the uninterrupted operation of process equipment [7, 8].

Increased requirements are also imposed on operating systems, on embedded operating systems [9]. Real-time operating systems (RTOS) are essential for computer systems that interact with objects external to the system. At the same time, the flow of events from these external objects requires algorithmic and software processing, as a rule, in a short period of time or during a time-limited interaction session. A number of examples of the

* Corresponding author: bahtiyar1975@mail.ru

implementation of such applications can be found in [10-12]. Increased performance requirements for parallel computing on multiple processors are now commonplace in automation systems in industry, energy, telemedicine, smart agriculture, etc.

And although modern RTOS are high-speed systems on parallel processors, for which the processing time can be less than tenths of a second, the time limits for such systems when processing external messages are critical. The external computing periphery defines strict regulations when time limits are specified for processing, within which it must be performed. Otherwise, the result of data processing will lose its relevance and, moreover, will lead to incorrect operation of the system or to its collapse. This is important for systems that are critical in terms of operational reliability. Many of these systems are implemented in fault-tolerant versions based on approaches with redundant versions of programs that are executed in parallel on a distributed multiprocessor system. For example, multiversion fault-tolerant software requires parallel execution of programs on several processors simultaneously, increasing the load on processors and memory, and also requiring efficient organization of interprocessor interaction [10].

Graphic processors have a significant advantage over other computing devices in the form of a large number of scalar processors combined into multiprocessors [13-17]. But this division is also a kind of disadvantage when creating parallel algorithms for the GPU, since it is not possible to use all the scalar processors of all multiprocessors while operating simultaneously on a single shared memory array [18]. Therefore, when developing parallel algorithms for the GPU, it is necessary to divide the algorithm into stages, each of which can be executed on a set of independent multiprocessors. This, in turn, can lead to the use of only one multiprocessor, which will reduce the increase in GPU computing performance to a minimum. Also, a significant role in increasing the performance of calculations is played by the amount of initial data that needs to be processed on the GPU.

Thus, when developing parallel algorithms for GPUs using the proposed model, you can use the following technique.

1. Select the main stages of the algorithm, taking into account the conditions presented in [18]. Due to the fact that this algorithm can be executed on different GPUs, the steps must be selected based on the parameters M_{GPU} and M_C . Those the number of these stages may vary depending on the amount of data being processed and the amount of global GPU memory and constant memory of a single GPU:

$$B = f(N, M_{GPU}, M_C). \quad (1)$$

2. Develop parallel algorithms for processing sections of initial data using one abstract graphical multiprocessor (AGM) for each stage. Having developed an algorithm for AGM, it is possible to scale it to all GPU multiprocessors, each of which will process its own section of the initial data at each stage. Due to the fact that the AGM model was developed taking into account the already known PRAM model, one of the main advantages of this model over others is the ability to use existing algorithms for the PRAM machine. Thus, each stage of the parallel algorithm for the GPU is an array of independent simultaneously operating PRAM machines that process their own sections of the source data.

These two steps are enough to develop a parallel algorithm for the GPU. The consequence of this technique are the criteria by which you can find out whether it is possible to transfer an already known parallel algorithm to a graphics processor:

- the parallel algorithm can be divided into stages, the number of which is determined by formula (1);
- each stage can be implemented using one AGM, which sequentially processes all the data of the stage.

Taking into account formula (1), the parallel algorithm for the GPU becomes independent of the hardware capabilities of the system on which it runs. Those. the number of stages and the execution time of each of them is calculated in real time depending on the model

parameters that must be determined before starting the program. At the same time, if several parallel algorithms for data processing on AGM are created, then based on the analysis and evaluation of the parameters of the algorithms, it is possible to choose the best algorithm for AGM in real time or leave the calculations on the CPU. After initialization of the model, the number of stages is found, and the algorithm starts to execute the stages, starting from the first one.

2 Materials and methods

For a particular computing system, the execution of parallel programs depends on many factors. This also applies to GPUs. To determine what specifically affects the performance of the algorithm, it is necessary to analyze these factors and identify the bottlenecks of the algorithm, which just contribute to lowering its performance.

It is proposed to use known methods for analyzing parallel algorithms for the PRAM model when analyzing a parallel algorithm implemented on a graphical multiprocessor. As a rule, these methods are based on the method of estimating the working complexity of the algorithm or on the method of stepwise complexity of the algorithm.

Based on the fact that the graphics multiprocessor physically has a finite number of scalar processors, we can programmatically emulate a larger number of them. Thus, you can organize the serialization of task execution. Considering formula (2) for graphic multiprocessors, it should be understood that there are special costs for the operational complexity of the algorithm. In this case, the totality of these costs should be commensurate with the costs that the system will incur for organizing access to the global memory of the GPU.

Therefore, it is very important to control the access of the algorithm to the global memory of the GPU and to be able to estimate the cost of organizing this access. However, global memory access and computational operations can overlap, lowering the overall runtime cost of the algorithm. The model does not reflect this possibility of overlap, but estimates an upper bound on the execution time of the algorithm on the AGM. Hence, it can be concluded that in parallel AGM algorithms, it is necessary to reduce the algorithm parameters associated with the overlap of computational operations and global memory access operations by reducing, first of all, the cycles spent by the multiprocessor on the implementation of global memory access operations.

Thus, within the framework of the proposed AGM model, it is possible to present the analysis of the algorithm stages and parameters at each step in the following form:

$$T_M(N,p) = O\left(\frac{W(N)+R(N)}{p} \cdot ceil\left(\frac{p}{p_{warp}}\right) + S(N)\right). \quad (2)$$

The main advantage of the proposed model is the ability to estimate the execution time of parallel computations on a particular GPU, taking into account its essential characteristics and algorithm parameters. When using well-known methods for estimating the execution time of a sequential version of the algorithm on the central processor, this makes it possible to obtain an estimate of the acceleration of calculations using GPU. Based on this acceleration, we can conclude that it is expedient to transfer computing to the GPU. Given the limited number of model parameters that can be found in real time, it is possible to make a decision about the target computing system (CPU or GPU) while the program is running.

3 Results and discussion

What matters to us is that a thread is the smallest computational unit in CUDA. He who runs on a scalar processor. The connection of this thread with the processor in the AGM or with

one PRAM processor of the machine is obvious. A set of threads is combined into a computing unit. This computational block is executed independently of other blocks. Moreover, this execution is carried out on a separate multiprocessor. In turn, AGM is developed on the basis of real graphic multiprocessors. This results in a set of threads within a compute unit also being associated with the AGM.

The processing of one block is a "closed process", that is, the interaction of blocks is excluded, and there is no possibility to change the calculated data in one block using another block. This is embedded in the development of the core of parallel computing in CUDA. Kernel development is the implementation of the PRAM algorithm for AGM. Kernel development implies strict consideration of the block index and the index of each processor in this block. It follows that the number of threads in a block must be equal to the number of processors in the PRAM algorithm. In this case, the shared memory of the multiprocessor becomes shared memory for them. It is important that in this case the copying of data from the global memory of the GPU to the shared memory of the multiprocessor and vice versa is organized. This provides the core of parallel computing in CUDA.

This type of operations is provided for by the proposed model. The number of blocks in CUDA is not limited and depends on the amount of data being processed, which must be presented in the form of a grid that unites all computing blocks. This concept of dividing data into blocks corresponds to dividing the processed data into AGMs, taking into account their independence from each other. In CUDA, all blocks are processed sequentially depending on the number of multiprocessors available.

This factor is taken into account when estimating the time execution of the parallel algorithm on the GPU. The proposed model of parallel computing on GPUs has a number of advantages over the CUDA model:

- allows you to estimate the execution time of a parallel algorithm depending on a particular GPU, taking into account only its essential features;
- simplifies the development of a parallel algorithm for a GPU using the already known PRAM parallel computing model, thereby hiding from the programmer the nuances associated with data parallelism.

4 Conclusions

Unlike OpenCL, for automation in agriculture the proposed parallel computing model is not common to all parallel computing devices, but is intended only for the CPU-GPU system. Accordingly, it does not consider job parallelism. Therefore, the proposed model can be considered a subset of the OpenCL model. The minimum computational unit in OpenCL is a work item that can be associated with one processor in an AGM (or one PRAM processor on a machine). Unlike OpenCL, the proposed model does not have private memory for each work item, which greatly simplifies the development of the program, relying on ready-made programming tools that solve issues with registers, addressing, etc. Naturally, the final program should fit in a limited amount of private memory in terms of volume for specific GPU.

One OpenCL computational block must be associated with a data block that can be processed on one AGM. These blocks must be independent of each other so that their calculation can be scaled to many AGMs. The local memory of each computing unit is shared memory for the AGM. Also in OpenCL, as well as in the proposed model, from the computational unit there is access to the global memory of the GPU and the memory of constants. Unlike OpenCL, in the proposed model, these two types of memory are separated and caching of various levels is not taken into account, which makes it possible to simplify the estimation of the execution time of a parallel algorithm in the form of an upper limit on

the execution time (a cache miss is always assumed when accessing the global memory of the GPU, constant memory - cache hit).

Thus, the proposed parallel computing model is more simplified compared to OpenCL and more adapted to the CPU-GPU system, and also has the same advantages as over the CUDA model.

Let's formulate these advantages.

1. The proposed AGM model based on the existing PRAM model allows developing, analyzing and comparing parallel algorithms for real graphic multiprocessors based on existing methods for the PRAM machine.

2. The proposed model of the GPU allows you to analyze and compare the various stages of the parallel algorithm for the GPU, which are based on PRAM algorithms for AGM.

3. The proposed model of parallel computing on GPUs allows you to create, analyze and compare parallel algorithms for GPUs in the CPU-GPU system.

The proposed model of parallel computing on GPUs is much simpler than existing models and has a number of advantages over them:

- highlights only the essential features of graphic processors that affect the performance of parallel computing;
- allows you to estimate the execution time of a parallel algorithm depending on a particular GPU, taking into account these features;
- simplifies the development of a parallel algorithm for a GPU using the already known PRAM parallel computing model, thereby hiding from the programmer the nuances associated with data parallelism;
- allows you to make a decision about the target computing system (CPU or GPU).

References

1. D. Kovalev, T. Mansurova, Y. Tynchenko, Modern Innovations, Systems and Technologies **1(2)**, 46-63 (2021). https://doi.org/10.47813/2782-2818-2021-1-2-46-63
2. A. Voroshilova, A. Kuznetsov, Modern Innovations, Systems and Technologies **1(2)**, 1-21 (2021). https://doi.org/10.47813/2782-2818-2021-1-2-1-21
3. A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, O. O. Storaasli, Scientific Programming **18**, 1-33 (2010)
4. A. V. Drozdov, Informatics. Economics. Management **1(1),** 0201-0216 (2022). https://doi.org/10.47813/2782-5280-2022-1-1-0201-0216
5. P. Kuznetsov, Y. Tynchenko, V. Kolesnik, Informatics. Economics. Management **1(1),** 0217-0228 (2022). https://doi.org/10.47813/2782-5280-2022-1-1-0217-0228
6. M. Kovito, Informatics. Economics. Management **1(2)**, 0121-0133 (2022). https://doi.org/10.47813/2782-5280-2022-1-2-0121-0133
7. G. Ishankhodjayev, M. Sultanov, B. Nurmamedov, Modern Innovations, Systems and Technologies **2(2)**, 0251-0263 (2022). https://doi.org/10.47813/2782-2818-2022-2-2-0251-0263
8. I. Baranov, Modern Innovations, Systems and Technologies **2(3),** 0139-0149 (2022). https://doi.org/10.47813/2782-2818-2022-2-3-0139-0149
9. E. Tanenbaum, *Modern operating systems* (Peter, SPb., 2002)
10. D. Gruzenkin, D. Shavarin, Modern Innovations, Systems and Technologies **2(3)**, 0127-0138 (2022). https://doi.org/10.47813/2782-2818-2022-2-3-0127-0138
11. O. R. Allaberganov, B. S. Rakhimov, S. K. Sobirova, F. B. Rakhimova, A. B. Saidov, AIP Conference Proceedings **2647**, 050025 (2022)

12. B. S. Rakhimov, M. S. Mekhmanov, B. G. Bekchanov. J Phys Conf Ser. **1889(2)**, 022090 (2021). https://doi.org/10.1088/1742-6596/1889/2/022090
13. B. S. Rakhimov, F. B. Rakhimova, S. K. Sobirova, J Phys Conf Ser. **1889(2)**, 022028 (2021). https://doi.org/10.1088/1742-6596/1889/2/022028
14. A. P. Karpenko, S. K. Chernov, Mechanical Engineering and Computer Technologies **1**, 185-214 (2013)
15. D. A. Forsyth, J. Pons, *Computer vision. Modern approach* (Publishing house Williams, M., 2004)
16. R. B. Saidovich, S. A. Bakhtiyarovich, B. B. Farkhodovich, Texas Journal of Medical Science **7**, 105-110 (2022)
17. P. P. Kudryashov, *Algorithms for detecting a human face for solving applied problems of image analysis and processing* (Moscow, 2007)
18. B. Rakhimov, O. Ismoilov, AIP Conference Proceedings **2432**, 060031 (2022). https://doi.org/10.1063/5.0089711