JMGCB

ISSN: 3033-1085

https://doi.org/10.61796/jmgcb.v1i1.143

Analysis Database Systems and Solve Medical Problems

Rakhimov Bakhtiyar Saidovich
Head of the Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan

Rakhimova Feroza Bakhtiyarovna
Senior teachers of the Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan

Saidova Zarina Bakhtiyar Qizi
Student 1 – course Urgench branch of Tashkent University of Information Technologies named after Muhammad al Khwarizmi, Uzbekistan

Received: Nov 20, 2023; Accepted: Des 21, 2023; Published: Jan 22, 2024;

Abstract: Due to this mechanism, threshold synchronization is achieved with a minimum amount of time. It is usually designed for communication between scalar processors via shared memory. The main function of graphics processors since their inception has been graphics processing. Subsequently, after it became possible to program the processing of model vertices and pixels of rendered three-dimensional scenes using special programs (shaders), the architecture of graphic processors changed significantly. After the advent of the first general-purpose programmable graphics it became possible to process commands not only for graphic data in vector form, but also to perform ordinary calculations for arbitrary data on a variety of special cores, while implementing data parallelism.

Keywords: Grafic processors, algorithm, memory, hardware, software.

Introduction.

The multilevel memory hierarchy allows access to global data through the first and second level caches. The second level of the cache is shared between data and texture units, while the first level is shared between the two multiprocessors and is intended for data only. When calculating on a multiprocessor, in addition to data from global memory, two additional types of memory are used: constant memory and shared memory. The constant memory is read-only. It is stored in video memory and cached on the 8 KB multiprocessor (the total size for all multiprocessors is limited to 64
KB; the constant memory is the same for all multiprocessors). The latency when accessing it can be the same time as for accessing global memory in case of a cache miss, but if there is a cache hit, then the access will be performed in 2 multiprocessor clock cycles [5]. Shared memory is intended for reading / writing and is organized in the form of memory banks (16 or 32), each of which can be accessed in 2 clock cycles. The entire bundle gets the request to access the shared memory[11]. In this case, requests may arise immediately to one bank of shared memory, which will entail a conflict and ordering of requests into the queue with an increase in the access time to bank data for the entire bundle. Therefore, it is important to take into account the coherence of requests when creating algorithms [1]. The amount of shared memory is also limited (16 KB), but in later versions of GPUs
(Compute Compatibility 2.x) it can be configured depending on the task. In the process of studying various types of access to memory caches, it was revealed that the required number of cycles can vary significantly [10].

The device that is the main one in the computing system (CPU) is called the Host. It runs the main sequential program, which transfers control to the parallel computing device Device (GPU) to

implement parallel computations. The program that Device runs is called the Kernel. The kernel is developed in the same language in which the sequential program (C / C ++) is implemented using special language additions. Parallel execution on a Device is implemented due to threads combined into blocks. The blocks, in turn, are combined into a (Grid) section, which must completely cover the data processed by the kernel. In order for the Device to process any data, it is necessary to transfer it to the Device memory, then get the result by copying the data in the opposite direction.

CUDA allocates five types of memory. These are registers, local, global, shared, constant and texture memory. Whenever possible, the compiler tries to place all local function variables in registers. These variables are accessed as quickly as possible. In the current architecture, 8192 32-bit registers are available per multiprocessor. In order to determine how many registers are available to one thread, it is necessary to divide this number (8192) by the block size. With the usual division into 64 threads per block, only 128 registers are obtained. Local memory, when the local data of procedures is too large, or the compiler cannot calculate some constant step for them when accessing, it can place them in local memory. This can be facilitated, for example, by casting pointers for types of different sizes. The CUDA documentation lists the ability to arbitrarily address global memory as one of the main technology advances. That is, you can read from any memory cell, and you can also write to an arbitrary cell (this is usually not the case on a GPU). Global memory is not cached. It works very slowly, the number of calls to the global memory should be minimized in any case. Global memory is mainly needed to store the results of the program before sending them to the host (in conventional DRAM). The reason for this is that global memory is the only kind of memory that can be written to. Shared memory is non-cacheable but fast memory. It is recommended to use it as a managed cache. Only 16KB of shared memory is available per multiprocessor. Dividing this number by the number of tasks in the block, we get the maximum amount of shared memory available per thread. Constant memory is cached, the cache exists in a single copy for one multiprocessor, which means that it is common for all tasks within the block. Constant memory is very easy to use. You can place any type of data in it and read it using a simple assignment. Texture memory is cached. There is only one cache for each multiprocessor, which means that this cache is common for all tasks within the block. Texture memory is not physically separated from global memory.

In PRAM, all control is performed using a single clock counter, and it is considered that all processors execute instructions synchronously with this counter. In one cycle, three actions are performed at once: reading data from the shared memory, performing an operation on the read data, and writing the results to the shared memory. This condition is met even if all processors perform different operations or a different number of memory accesses. That is why this model is idealized, because in real computers, these actions vary in time. Nevertheless, this model is suitable for creating, analyzing and comparing algorithms, taking into account the following assumptions [6]:

1) the number of processors in the machine is not limited;
2) each processor has equal access to any cell of the shared memory;
3) the size of the shared memory is not limited;
4) there is no competition for resources;
5) processors operate in MIMD mode.

Objective Statement

Obviously, not all computer vision algorithms can be parallelized on GPUs. Any artificial computer vision system, regardless of its area of application, should include the following typical stages of work:

1) image acquisition (photo or video filming);
2) preliminary processing;
3) highlighting characteristic features;

4) detection or segmentation;

5) high-level processing.

One of the basic features of orthogonal bases is presence of fast algorithms for definition of spectral factors. Fast algorithms allow to reduce quantity of arithmetic operations and volume of necessary memory. The increase in speed is as a result reached at use of orthogonal bases for digital processing signals [1, 2, 3, 4].

Materials nad Methods Bulk Synchronous Parallel (BSP, Valiant, 1990) is an extension of the PRAM model [8]. Later it was transformed into a parallel computing standard [6]. In this model, the main attention is paid to communication and synchronization of processors. Therefore, the model consists of the following components [7]:

1) Processors, each of which has fast local memory and can execute multiple virtual threads;

2) A communication network that allows sending and receiving communication messages from processors;

3) A mechanism for synchronizing all processors at certain points in time.

The algorithm in BSP has a vertical and horizontal structure [10]. The vertical structure is a sequence of supersteps, each of which consists of three main steps:

1) Local calculations on each processor using only the data that was stored in the processor's local memory. Calculations are performed asynchronously;

2) Communication between processes using a communication network (messaging);

3) Barrier synchronization of processes. Once a process reaches a synchronization point (a barrier), it suspends computation and waits for other processes to reach that synchronization point. the results of the program before sending them to the host (in conventional DRAM).

Unlike the G80, the multiprocessor in the R600 does not have a local shared memory, but consists of a certain number of vector processors. It is the number of vector processors that determines the size of a bunch of processes (wavefront - in ATI's terminology). In the best configuration, one multiprocessor included 16 vector ones. Each vector processor consists of four scalar processors, one transcendental function processor, a branch block and general purpose registers. Thus, at one time, due to the use of VLIW, five operations on 32-bit numbers can be simultaneously performed on one vector processor. But most general-purpose parallel computing emphasizes scalar computing, so this parallelization feature is rarely used. Different VLIW commands can be executed on all multiprocessors, but on one multiprocessor for all vector processors, the instruction must be the same, but with different address registers. Access to video memory is performed for all threads simultaneously within 300 to 600 multiprocessor cycles [6], but due to the use of VLIW and scheduling of thread bundles, the delay in accessing the global memory of the video adapter is effectively hidden.

The memory controller reports to the task manager. Processed data and constant memory are stored in video memory and, if necessary, cached in caches of the second and first levels, while they are common to all multiprocessors.

Results and Discussion

The models of parallel programming discussed above are intended primarily for the programmer to have an idea about the main structural elements of graphic processors, which include memory and computing elements, and the relationships between them. In addition, both CUDA and OpenCL have some algorithm abstraction that assumes that input and output data are represented as an array of elements, each of which is processed independently of each other, due to which data parallelism is achieved. We highlight the main disadvantages of these models:

1) there is no mathematical description of the abstract model of the GPU, so it is impossible to estimate the running time of a particular algorithm on different GPUs;
2) significant parameters of parallel algorithms have not been identified, thanks to which it is possible to analyze and compare these algorithms in terms of execution time on a GPU with a certain configuration;
3) despite the fact that the models are heterogeneous (i.e. they take into account not only graphics processors, but also central ones), they do not have methods for making a decision about the target computing system;
4) for these models, general principles for optimizing parallel computing have not been developed (in [2, 6], optimization methods are given for a specific platform, but not for a model);
5) there is no general methodology for the development of parallel algorithms.

These shortcomings are associated with a number of factors that one has to face when developing a parallel computing model [9], the main of which is the lack of a formal parallel computing model using a central and graphic processor. A formal model is provided by all abstract models of parallel computing with their abstract machines, but it is quite difficult to find a formal model suitable for analyzing parallel computing on GPUs due to the diversity of their architectures. Nevertheless, there are attempts to formalize some aspects of parallel computing on GPUs.

Conclusions

Computer vision as a scientific discipline refers to theories and technologies for creating artificial systems that receive information from an image. Despite the fact that this discipline is quite young, its results have penetrated almost all spheres of life. Computer vision is closely related to other practical areas [3,14]: 1) image processing, the input data of which are two-dimensional images obtained from a camera or artificially created. This form of image transformation is aimed at noise suppression, filtering, color correction, etc.; 2) image analysis, which allows obtaining certain information directly from the processed image. Such information may include the search for objects, characteristic points, segments, etc.; 3) vision of the robot, designed to orient the robot in space by modeling the environment from images received from video cameras; 4) machine vision, which is used in production and industry for automatic product quality control, product defect detection, measurement control, etc. Typical applied problems of computer vision are [3,4]: 1) Detection of objects in the image. Despite the fact that a person can easily select specific objects from an image, this problem has not yet been completely solved for artificial systems. Most of the solutions are of a particular nature, based on the specific properties of the object being searched for, and, accordingly, are not suitable for searching for objects that do not have them. There are several universal algorithms for object detection (neural networks, Viola-Jones, etc. [8]), which are slow and have a serious detection error with slight deviations of objects from the desired ones specified during training, but, nevertheless, their simplified versions widely used for small images. Therefore, an important task while maintaining the accuracy of detection is to speed up calculations [5]; 2) Recognition of objects in the image [1]. This task is a continuation of the previous one, the result of which is an array of areas where objects can be found. The purpose of this task is to determine the presence in these areas of a specific class of objects that already has more specific features and, accordingly, can be better classified; 3) Identification of objects, the result of which can be a conclusion about the correspondence of the recognized object to a specific (unique) instance (for example, a fingerprint, the face of a specific person, a car number, etc.). Separately, it is worth highlighting from this group character recognition systems, the accuracy of identification of which affects the quality of the recognized material; 4) Search for images in the database by content, based on the recognition of a particular class of objects. In this case, the performance of the artificial system plays a significant role in speeding up the search for images; therefore, the possibility of parallelizing the algorithms of this group of tasks is considered very important; 5) Reconstruction of a three-dimensional scene from a certain set of input images (video stream) allows you to determine the positions of objects and a

source in three-dimensional space, used to move robots, create panoramic images, etc.; 6) Tracking of moving objects in a video stream provides for direct determination of the position of an object in space by changing its position on two-dimensional images while maintaining the characteristic features of the object. This task is very resource-intensive and must be performed in real time, so the main emphasis when creating algorithms for this area is on their performance; 7) Image processing. This task area is designed to transform the pixels of two-dimensional images and is a priority for other computer vision tasks. Almost all transformations are filtering transformations, i.e. a set of operations is performed on each pixel of the image, depending on other pixels that are in close proximity to the desired one using special matrices.

References

1. P. P. Kudryashov Algorithms for detecting a human face for solving applied problems of image analysis and processing: author. dis. Cand. tech. Sciences: 05.13.01. - M, 2007.
2. Tanenbaum E. Modern operating systems. 2nd ed. - SPb .: Peter, 2002 .-- 1040 p .: ill.
3. Forsyth DA, Pons, J. Computer vision. Modern approach / D.A. Forsyth, J. Pons: Trans. from English - M .: Publishing house "Williams", 2004. - 928 p .: ill. - Parallel. tit. English
4. Frolov V. Solution of systems of linear algebraic equations by the preconditioning method on graphic processor devices
5. Brodtkorb A.R., Dyken C., Hagen T.R., Hjelmervik J.M., Storaasli O.O. State-of-the-art in heterogeneous computing / A.R. Brodtkorb, C. Dyken, T.R. Hagen, J.M. Hjelmervik, O.O. Storaasli // Scientific Programming, T. 18, 2010. - S. 1-33.
6. Rakhimov, BS; ,Russian “Information technologies in medical education”,METHODS OF SCIENCE Scientific and practical journal,12,25-7,2017,
7. Rakhimov, BS; Ismoilov, OI; Ozodov, RO; ,Russian “Software and automation of forensic examination”,METHODS OF SCIENCE Scientific and practical journal,11,28-30,2017,
8. Rakhimov, Bakhtiyar S; Khalikova, Gulnora T; Allaberganov, Odilbek R; Saidov, Atabek B; ,Overview of graphic processor architectures in data base problems,AIP Conference Proceedings,2467,1,020041,2022,AIP Publishing LLC
9. Saidovich, Rakhimov Bakhtiyar; Kabulovna, Sobirova Sabokhat; Bakhtiyarovna, Rakhimova Feroza; Akbarovna, Allayarova Asal; Bakhtiyarovich, Saidov Atabek; ,ANALYSIS OF THE GRAPHICS PROCESSORS FOR MEDICAL PROBLEMS,PEDAGOGS jurnali,11,4,167-177,2022,
10. Allaberganov, Odilbek R; Rakhimov, Bakhtiyar S; Sobirova, Sabokhat K; Rakhimova, Feroza B; Saidov, Atabek B; ,Problem for medical system with infinite zone potential in the half line,AIP Conference Proceedings,2647,1,050025,2022,AIP Publishing LLC
11. Rakhimov, Bakhtiyar Saidovich; Saidov, Atabek Bakhtiyarovich; Shamuratova, Inabat Ismailovna; Ibodullaeva, Zarnigor Ollayor Qizi; ,Architecture Processors in Data Base Medical Problems,International Journal on Orange Technologies,4,10,87-90,2022,Research Parks Publishing
12. Rakhimov, Bakhtiyar Saidovich; Saidov, Atabek Bakhtiyarovich; Allayarova, Asal Akbarovna; ,Using the Model in Cuda and Opencl for Medical Signals,International Journal on Orange Technologies,4,10,84-86,2022,Research Parks Publishing
13. Saidovich, Rakhimov Bakhtiyar; Musaevich, Yakubov Durumboy; Bakhtiyarovich, Saidov Atabek; Qizi, Saidova Zarina Bakhtiyar; ,ANALYSIS OF THE DEVICE FEATURES OF GENERAL-PURPOSE PROGRAMMABLE GRAPHICS PROCESSORS,CENTRAL ASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES,4,1,100-103,2023,
14. Рахимов, Бахтияр Саидович; Жуманиёзов, Сардор Пирназарович; ,Аппаратно-ориентированный алгоритм вычисления коэффициентов в базисах J-функций,Актуальные вопросы технических наук,59-62,2015,