ISSN NO: 2770-2936
Date of Publication:25-04-2022

Analysis And Using of the Features Graphics Processors for Medical Problems

Rakhimov Bakhtiyar Saidovich[1], Saidov Atabek Bakhtiyarovich [2], Babajanov Boburbek Farkhodovich [3], Karimov Doston Alisher Ugli[4], Musaeva Mukhtasar Zayirjon Qizi[5]
[1]Head of the Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan, bahtiyar1975@mail.ru
[2,3,4,5]students 3 – course Urgench branch of Tashkent University of Information Technologies named after Muhammad al Khwarizmi, Uzbekistan, ootabek2001@mail.ru, babajonovboburbek@gmail.com, karimovdoston7011@gmail.com, mukhtasar@gmail.com

Abstract. 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, the architecture of graphic processors changed significantly. After the advent of the first general-purpose programmable graphics processors, 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. Therefore, Graphics Processing Units show high efficiency rates when parallelizing programs that process a lot of data of the same type. Such programs include shaders: a vertex shader processes 3D vertices with different parameters, a pixel shader processes 2D pixels on the screen using interpolated data. Even before the advent of general-purpose kernels, there were attempts to simulate the processing of arbitrary data of the same type written into textures using pixel shaders, which, of course, gave a performance boost compared to the CPU. To develop an abstract model of the GPU, consider the general structure of modern GPUs, as well as models designed for their programming.

Key words: Graphics Processing Units, pixels, architecture of graphic processors, processor.

Introduction.

PRAM (Parallel Random Access Machine) is an idealized abstract model of a synchronous shared memory machine. This model was proposed in 1978 by S. Fortune and J. Wyllie [5] to estimate the performance (in particular, the execution time) of parallel algorithms.

PRAM assumes the presence of an infinite number of processors, each of which can synchronously execute various instructions on data in a shared memory, thus operating in MIMD mode [2]. If all processors execute the same instructions, then this model can be considered an abstract SIMD machine. The PRAM model does not consider the problems of synchronization and communication of processors, and the main instructions that processors can execute are the same as those of a conventional RAM (Random Access Machine, a machine with random access to memory) [9]: arithmetic, logical operations, and memory access operations . The latter cause a number of problems in parallel algorithms, therefore, within the framework of the PRAM model, several different strategies have been developed to resolve memory access conflicts:

1) Concurrent Read, Concurrent Write (CRCW, simultaneous reading with simultaneous writing) - at any time, data from the shared memory can be read and written in parallel by any processors;

2) Concurrent Read, Exclusive Write (CREW, simultaneous reading with an exclusive write) - at any time, data from the shared memory can be read in parallel by any processors, but writing can be done by only one processor;

3) Exclusive Read, Concurrent Write (CRCW, exclusive reading during simultaneous writing) - at any time, data from the shared memory can be read by only one processor, and written in parallel by several processors;

4) Exclusive Read, Exclusive Write (CRCW, exclusive read with exclusive write) - at any time, data from the shared memory can be read and written by only one processor.

ISSN NO: 2770-2936
Date of Publication:25-04-2022

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 [96]:
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.
To simulate algorithms on a PRAM machine, emulators were created that reflect the main features of this model [9].
To model and analyze algorithms on PRAM, it is necessary to represent their calculations in the form of an acyclic directed graph “operations-operands” [1]:
$G = (V, R)$
V where is the set of graph vertices representing the operations of the algorithm;
R is the set of graph arcs.
It is obvious that the number of different variations of calculation schemes for the constructed graphs will have different possibilities for parallelization. Operations between which there is no path in the constructed graph can be performed in parallel. Thus, the next step to parallelize the algorithm is to build a schedule for the required number of processors:
$H(p) = \{(i, P_i, t_i) : i \in V\}$

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 [1]. 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).
The entire R600 architecture for parallel computing can be divided into three blocks: a flow control block, an array of vector processors, and a memory controller.
The flow control block receives commands from the central processor to perform parallel processing of data and generates a schedule for the execution of threads. It consists of a command processor, a thread generator, and a command dispatcher. The command processor generates interrupts for the CPU during the execution of parallel computations and gives the job to the thread generator. Based on the information received, the thread generator compiles a schedule for blocks of threads executed on multiprocessors and converts it into an array of VLIW (very long instruction word, very long machine instruction [7]) instructions. Thus, all calculations on the graphics adapter are performed according to the EPIC architecture (Explicitly Parallel Instruction Computing, a system of instructions with explicit parallelism [6]). This

ISSN NO: 2770-2936
Date of Publication:25-04-2022

architecture allows you to schedule the load of all processors on each cycle [4]. In this case, the schedule is compiled statically with the most efficient loading of multiprocessors. The commands are then passed to the command dispatcher for execution, which has access to both the array of vector processors and the memory controller in order to execute the schedule correctly.

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.

The next architecture of R700 chips appeared in June 2008 [6, 8]. The main differences from the R600 were as follows:

1) multiprocessors now have local shared memory (16 Kb);
2) a separate first-level cache for data appeared for each multiprocessor;
3) the cache of constants and instructions is separated from the data cache;
4) the second-level data cache is divided into several memory controllers;
5) the number of multiprocessors increased to 10 in the best configuration.

Thus, in this architecture, it became possible to cache data in the local memory of the multiprocessor, allowing exchange operations between the processors of the multiprocessor to be carried out faster than through video memory.

The architecture of the latest generation R800, presented in September 2009 [7], is almost identical to the R700. The changes mainly affected the quantitative characteristics:

1) the size of the shared memory of the multiprocessor has been increased to 32 KB;
2) increased the number of multiprocessors to 20 in the best configuration.

ATI Stream SDK and the OpenCL language, which uses its own heterogeneous model [5, 8], are used to program ATI GPUs. Unlike CUDA, this model is applicable not only to data parallelism, but also to task parallelism, and takes into account the number of CPU cores. OpenCL is based on a C-like language for writing program cores.

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;

ISSN NO: 2770-2936
Date of Publication:25-04-2022

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.
The first analytical model of parallel computing on GPUs, described in [3] by S.Hong and H.Kim, focuses on accessing various types of memory and the associated delays. In this model, two metrics are introduced: MWP (Memory Warp Parallelism, parallel access of beams to memory) and CWP (Computation Warp Parallelism, parallel processing of beams). MWP denotes the number of memory request bundles that can be executed by the multiprocessor immediately after a memory access request appeared in the original bundle, before the request is processed, and therefore the next bundle instruction can be executed. CWP denotes the number of bundles that the multiprocessor can compute while processing one memory access request. 1 is usually added to the CWP to account for the current request bundle. The MWP parameter plays a significant role in this model, because it allows you to "hide" the delay when accessing memory by performing the same requests throughout the access. This parameter is determined by the memory bandwidth, memory access model and the number of active bundles on the multiprocessor. The CWP parameter is already determined by the program itself and depends on the number of instructions in one bundle. The larger it is, the fewer calculations are performed when the beam accesses memory, which entails downtime for multiprocessors. This model makes it possible to calculate the MWP and CWP data (formulas are given in) based on the generated algorithm code and GPU parameters only for the CUDA programming model, and then select the necessary program execution paths based on the decision tree. We highlight the main disadvantages of this model:
1) the model takes into account only GPUs based on NVIDIA chips, i.e. there is no abstract machine that allows you to calculate parameters for other GPUs;
2) the model is focused on the access of beams to memory and the computational intensity of these beams, while not allowing to estimate the execution time of the entire algorithm according to these parameters, depending on the initial data;
3) when developing parallel algorithms, it is quite difficult to operate with beams, because the main focus of the programmer is on individual processors;
4) the model does not take into account the memory hierarchy (and associated latency) in GPUs.
The analytical model of parallel computing on graphic processors has not received further development.
Attention was drawn to the memory hierarchy in the models of S.Byna, X.H.Sun [8]. The model is described by a set of parameters that define each level of the hierarchy and access to it. This model works for the CPU, but, nevertheless, there is a possibility of application for the GPU, if each parameter of the model is determined using special statistical testing programs given in [1]. However, due to the complexity of the model due to the large number of parameters, its application will cause difficulties when.

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]: 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 [34]: 1)

ISSN NO: 2770-2936
Date of Publication:25-04-2022

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. 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. 1-33Forsyth 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
2. Frolov V. Solution of systems of linear algebraic equations by the preconditioning method on graphic processor devices
3. 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.
4. Rakhimov, B.S., Rakhimova, F.B., Sobirova, S.K. *Modeling database management systems in medicine*, Journal of Physics: Conference Seriesthis link is disabled, 2021, 1889(2), 022028
5. Rakhimov, B.S., Mekhmanov, M.S., Bekchanov, B.G. *Parallel algorithms for the creation of medical database*, Journal of Physics: Conference Seriesthis link is disabled, 2021, 1889(2), 022090
6. Tanenbaum E. Modern operating systems. 2nd ed. - SPb .: Peter, 2002 .-- 1040 p .: ill.
7. Zaynidinov H., Mallayev O., Kuchkarov M. Parallel algorithm for modeling temperature fields using the splines method 2021 IEEE International IOT, Electronics and Mechatronics Conference, IEMTRONICS 2021 - Proceedings, 2021, 9422645
8. Zaynidinov H., Makhmudjanov S., Rajabov F., Singh D. IoT-Enabled Mobile Device for Electrogastrography Signal Processing Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2021, 12616 LNCS, стр. 346–356

ISSN NO: 2770-2936
Date of Publication:25-04-2022

9. Zaynidinov H.N., Yusupov I., Juraev J.U., Singh D. Digital Processing of Blood Image by Applying Two-Dimensional Haar Wavelets Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in bioinformatics), 2021, 12615 LNCS, стр. 83–94