Procedia of Engineering and Medical Sciences

Proceedings of the International Congress on "Medical Improvement and Natural Sciences" | 2022

Analysis Medical Problems with Database Systems and Create New Models

Rakhimov Bakhtiyar Saidovich [1]
Rakhimova Feroza Bakhtiyarovna [2]
Xusainov Shixnazar Madaminovich [3]
Saidov Atabek Bakhtiyarovich [4]

[1] *Head of the Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan*
[2] *Senior teacher of the Department of Biophysics and information technologies of Urgench branch of Tashkent Medical Academy, Uzbekistan*
[3] *master 2 – course Urgench branch of Tashkent University of Information Technologies named after Muhammad al Khwarizmi, Uzbekistan*
[4] *student 4 – course Urgench branch of Tashkent University of Information Technologies named after Muhammad al Khwarizmi, Uzbekistan*

Annotation. 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 processors (G80 architecture from NVIDIA, R600 architecture from ATI), 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 (GPU) 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 (image matrices) 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: Grafic processors, algorithm, memory, hardware, software.

Introduction.

For the first time, the G80 architecture was presented in November 2006 and became widespread in several versions of graphic processors, differing in the number of multiprocessors installed on them [1]. If we remove all the elements of the architecture associated with the processing of graphics, we get the following structural diagram.

The main elements of the G80 architecture are:

1) A flow control block designed to generate a schedule and control the execution of flows. This block is controlled by the main processor of the system (CPU), which delegates a parallel task consisting of many threads to the GPU;
2) Computing unit, consisting of multiple streaming multiprocessors that receive and process parallel streams (work in MIMD mode);
3) Memory hierarchy, in which the main element is video memory (graphics adapter memory). It is accessed through L1 and L2 caching. However, the memory access operation is very expensive and can cost from 400 to 600 multiprocessor clock cycles.

The multiprocessor consists of the following components [4]:

1) 8 unified scalar processors (SP, Stream Processor), which allow performing both operations on integers and on floating point numbers;
2) 2 blocks for calculating transcendental functions with single precision (SFU, Super Function Unit);
3) shared memory block;
4) flow control unit
5) constant memory cache;
6) General purpose registers (RF, Register File).

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 next architectural solution for NVIDIA graphics processors was GT200, presented in June 2008 [9]. The main differences from the G80 in terms of general calculations are:

1) the number of TPCs (Thread Processing Cluster, multiprocessor clusters) has increased;
2) in each TPC, the number of multiprocessors has increased to 3 per cluster, while the L1 cache has also increased;
3) the number of registers for program instructions has doubled;
4) each multiprocessor has a block for computing operations on double precision floating point numbers.

Thus, this architecture made a breakthrough in the number of scalar processors on the graphics adapter and the amount of data processed, but the structural elements and their arrangement remained the same.

Unlike the GT200 architecture, the third generation Fermi architecture [8], presented in September 2009, has significantly redesigned the structural diagram of the GPU.

The central element of the architecture is the L2 cache, which is used to access video memory. Accordingly, its volume has also increased significantly. Streaming multiprocessors are formed around this

https://procedia.online/ ISSN-2795-563X

cache, which have also changed from previous architectures:

1) the number of scalar processors has increased to 32, moreover, they are optimized for working with 64-bit data;
2) it became possible to execute two competing bundles of streams on one multiprocessor;
3) the number of blocks for calculating transcendental functions has increased to 4;
4) it became possible to manage the amount of shared memory and the first-level cache for data;
5) there were blocks for calculating data addresses located in a single address space.

The heterogeneous CUDA (Compute Unified Device Architecture) model is used to program NVIDIA GPUs [3, 12]. CUDA includes a superscalar parallel computing programming model, as well as libraries and compiler extensions for several high-level languages. The general scheme of CUDA operation.

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.

Objective Statement

Consider the main groups of computer vision algorithms using data parallelism (this classification is a generalization of groups from):

1) image transformation algorithms - input and output data are two-dimensional images, the coordinates of the output image element differ from the coordinates of the input element. These algorithms include: affine transformations, coordinate system transformations, etc .;
2) filtering algorithms - input and output data are two-dimensional images. Each pixel in the output image is the result of an operation on a group of pixels in the input image that fall into a window of a certain size (filter). Filters can be represented by various mathematical devices: cellular automata, neural networks, functions, etc .;

3) statistical algorithms - input data are two-dimensional images, output data are arrays with statistical information. Each element of the original data can be used for statistical calculations if it meets the requirements of the algorithm. An example of such algorithms can be the calculation of histograms, Hook's transformation, grouping of elements, etc. Parallelization of such algorithms requires communication between processors to combine the accumulated statistical data;

4) recursive algorithms - input and output data are two-dimensional images. Each element of the original image contributes to the formation of all the image elements in the output. These algorithms include: calculation of the integral of the image, distance transformation, etc. This type of algorithms is very difficult to parallelize, since in most cases, computation of a single output item requires the result of previous input items.

One of the basic features of orthogonal bases is presence of fast algorithms for definition of spectral factors. Fast algorithms allow reducing 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).

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 architecture allows you to schedule the load of all processors on each cycle [4,12]. 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.

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

https://procedia.online/ ISSN-2795-563X

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

If the GPU is the Device, then the size of the partition and blocks is limited. Each block is executed on a separate multiprocessor independently of other blocks. Therefore, the width and height of the block are determined by the maximum number of simultaneously processed threads on the multiprocessor. In turn, when executed, the blocks are divided into bundles of 32 threads, which are launched in accordance with the commands of the multiprocessor thread control unit. Communication between threads in a block is done using shared memory and barrier thread synchronization. Series of numerical experiments have been carried out on the research of piecewise-quadratic bases. With use of the offered algorithm of calculation of coefficients in Haar and Harmut's piecewise-quadratic bases, "Table-1" is achieved. Here the factor of compression Kc is defined by the formula:

Kc=N/(N-N1),

Where N - Quantity of readout of function N1 - Quantity of the zero factors received as a result of use of offered algorithm.

The numerical experiments allow us to draw a conclusion that the number of zero coefficients at digital processing of signals received as a result of bench tests in Walsh and Haar's piecewise-quadratic bases ranges from 5 % up to 17 %, when processing the geophysical signals received as a result magnetic exploration we get values ranging from 5 % up to 25 %, and while processing elementary functions (and also functions consisting of their combinations) this parameter gives us value from 10 % up to 70 % with an accuracy of 10-4-10-6. It is established, that decomposition (4) allows receiving high speed in Haar's basis and the big factor of compression in Walsh's basis. Also as a result of researches it is revealed, that with increase in quantity of readout function N, the values of factors decreases on exponential law.

References

1. Rakhimov BS, Mekhmanov MS, Bekchanov BG. Parallel algorithms for the creation of medical database. J Phys Conf Ser. 2021;1889(2):022090. doi:10.1088/1742-6596/1889/2/022090
2. Rakhimov BS, Rakhimova FB, Sobirova SK. Modeling database management systems in medicine. J Phys Conf Ser. 2021;1889(2):022028. doi:10.1088/1742-6596/1889/2/022028
3. Rakhimov B, Ismoilov O. Management systems for modeling medical database. In: ; 2022:060031. doi:10.1063/5.0089711

4. Rakhimov BS, Khalikova GT, Allaberganov OR, Saidov AB. Overview of graphic processor architectures in data base problems. In: ; 2022:020041. doi:10.1063/5.0092848
5. 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.
6. Tanenbaum E. Modern operating systems. 2nd ed. - SPb .: Peter, 2002 .-- 1040 p .: ill.
7. 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
8. Frolov V. Solution of systems of linear algebraic equations by the preconditioning method on graphic processor devices
9. 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.
10. 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
11. 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
12. 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