V. V. RUSINOV, O. V. CHEREVATENKO, L. M. PUSTOVIT, O. M. PUSTOVIT, A. VOLOKYTA

ISOEFFICIENT CALCULATION METHOD FOR DISCRETE FOURIER TRANSFORM

The paper considers the issue of isoefficiency of MPP systems and heterogeneous CPU-GPU systems on the problem discrete Fourier transform. The development of parallel applications as its goal can have not only reduction of execution time, but also provision of opportunities to solve problems of greater dimensions. Feature parallelization of the algorithm includes the effective use of hardware when increasing the dimensionality of the problem an important characteristic of parallel computing.

Key words: isoefficiency, heterogeneous calculations, Fourier transform

Relevance of the research topic. The creation of iso-efficient systems allows you to deploy a system for some task taking into account its efficiency. The efficiency of parallel computing depends on the algorithm for implementing the task mapping on the system. The purpose of this article is to consider the process of creating an isoefficient system for solving practical problems using the Fourier transform as an example, put the algorithm in the MPP system and use an empirical approach based on machine learning to approximate the isoefficiency function for systems with different architectural solutions.

The task that will be performed on a parallel and heterogeneous system is the discrete Fourier transform algorithm. The discrete Fourier transform is used to solve problems of spectral analysis - to study a signal through its division into a set of signals. The nonlinear complexity of this algorithm is interesting from the point of view of parallel processing.

The isoefficiency function will be described based on the results of modeling the problem on the considered systems. For a parallel MPP system, the result can be obtained analytically. For a heterogeneous CPU-GPU system, it is currently impossible to obtain the isoefficiency function analytically, instead, an approach using machine learning algorithms will be applied.

Actual scientific researches and issues analysis. There are a number of scientific publications that study the subject of isoefficient systems based on the MPP architecture [2, 3]. Methods of scaling parallel algorithms for solving certain applied problems for their display on the MPP of a system with a given topological organization are considered. The approach of creating iso-efficient systems allows you to analyze the algorithm for its quality and parallelization capabilities on a given

topology and, as a result, the capabilities to scale the system with the expected efficiency of the task.

Most modern supercomputers and large data centers use systems that have both central processing units (CPUs) and graphics processing units (GPUs) on the nodes. General Purpose GPU Computing (GPGPU) has opened the way for PC users to experiment and work on projects that involve GPUs to handle labor-intensive tasks. Heterogeneous systems based on CPU and GPU have been widely researched and are quite a promising direction for the development of parallel computing. Despite the proliferation of CPU-GPU-based systems, to date isoefficiency has not been investigated with respect to heterogeneous systems.

The statement of basic materials. Isoefficient systems are systems with a given efficiency of solving problems, which, being described in advance, is constantly maintained. Let's consider the theoretical possibility of creating such systems.

The formula for the efficiency of parallel processing on a parallel system is as follows, determining the possibility of achieving the required efficiency of parallel computing systems:

$$E = \frac{S}{N} \quad (1)$$

By varying the parameters n, that is, the dimension of the problem, and N, the number of processors, it is possible to achieve a linear increase in productivity with an increase in the number of processors. This means that when performing computational processes, it is possible to determine in advance the necessary efficiency of their implementation [4].

The task completion time is predicted using "precedential" information - how long has it taken for the task of the same type to be completed. A task scheduler is created, which forms the initial result, which is simulated from the analysis of already completed tasks, then the model is used to predict the time of completion of new tasks. The scheduler analyzes the size of the data with which tasks work and their execution time, and also takes into account the time of previous tasks of the same type.

Using a similar deterministic correlation model, the execution time of a task of the same type is predicted, where the input parameter is the size of its data.

The problem that will be simulated on the MPP system and the heterogeneous CPU-GPU system is the Discrete Fourier Transform. The implementation of the discrete Fourier transform (DFT), which is the basis of spectral analysis, is an informal representation of signals, i.e. the investigated signals are represented by a sequence of counts x(k) .

$$F_x(p) = \sum_{k=0}^{N-1} x(k)e^{-jktp} \tag{2}$$

$$\rightarrow {}_p \rightarrow p \rightarrow p; \ = \frac{2}{T} \tag{3}$$

It can be seen from the formulas that the signal presentation intervals are equal to 2π, which is the period of low frequencies. To increase the accuracy, it is necessary to increase the interval T.

$$t \rightarrow t_k \rightarrow kt \rightarrow k; \ t = \frac{T}{N} = \frac{1}{k_{req}} f' \tag{4}$$

The DFT is a simple calculation procedure of the "matching" type, the estimate of its complexity is: N2 + N. To implement it, you need to calculate the turning coefficients of the DFT:

$$W_N^{pk} = e^{-jktp} \tag{5}$$

These rotation coefficients are recorded in the ROM, that is, they are constants.

$$W_N^{pk} = e^{-jk\frac{T}{N}p\frac{2}{T}} = e^{-j\frac{2}{N}pk} \tag{6}$$

In formula (4), W_N^{pk} (rotation coefficients) do not depend on T, but only on the dimension of the transformation N, therefore they are not presented in exponential form, but in trigonometric form

$$W_N^{pk} = \cos cos\left(\frac{2}{N}pk\right) - jsin\left(\frac{2}{N}pk\right) \tag{7}$$

The rotation coefficients are repeated, for this reason, the changes in values are described exactly to the specified values: p – up to (N-1), k – up to (N-1), with a period of N(2). When taking out the sign of the coefficient, only half of the coefficients can be stored. The real and imaginary parts of the coefficients are stored separately in the ROM [6].

In its general form, the DFT can be represented as follows:

$$F_x(p) = \sum_{k=0}^{N-1} x(k)W_N^{pk} \tag{8}$$

From such a formulaic definition, it is appropriate to present the DPF in the form of a graph.

CUDA (Compute Unified Device Architecture) is a parallel computing platform and API developed by Nvidia. It allows developers to use Nvidia graphics cards for general computing. The CUDA platform provides the developer with direct access to the system of video card commands and elements of parallel computing, for the execution of computing kernels (compute kernel).

The computing core is the main working unit, with the help of which the developer describes the algorithm. This term is not only used for GPUs, it is also used for FPGAs, TPUs, DSPs. For CUDA, the programming paradigm is very closely integrated with vector computing, based on the assumption that a kernel call is executed concurrently in a number of independent elements, allowing parallelism at the data level. However, there are also atomic operations that can be used to synchronize between elements. Each call receives indices for 1 or more dimensions, which are used for data addressing or buffering [7].

The architecture of the GPU belongs to the SIMD class, that is, data is sent to each core mentioned above, on which one operation is performed in one cycle. As an example, it is suggested to examine the TU102, which is the basis of the mainstream GPU RTX 2080Ti and in the professional GPU Quadro RTX 6000. It consists of 6 clusters of graphics processing, 36 clusters of texture processing and 72 Streaming Multiprocessors (SM). SM consists of 64 CUDA cores, 8 tensor cores, 256 kilobytes of register file, 4 Texture Units, 96 kilobytes of shared memory. Before that, each core has access to 6144 kilobytes of L2 cache.

Results. To study the efficiency of calculations, it is necessary to use metrics of the execution time of the algorithm sequentially (on one processor) and in parallel (on several processors). Based on the obtained time values, the effectiveness of the parallel algorithm can be investigated.

The first considered MPP system – a hypercube of degree 1. MPP (decoded as massive parallel processing) is a massively parallel architecture of computer systems. In this type of architecture, memory is physically separated. The system contains separate blocks (modules), inside which there are a processor, communication processors (routers), a local memory bank, network adapters, hard drives, input/output devices.

Only processors from the same module have access to the RAM of a separate node. Blocks are connected to each other by communication channels. It is possible for users to obtain the number of the processor and the processors to which it is connected, after which data exchange between them can be initiated.

The main advantages of systems with MPP architecture: good scalability, no need for clock synchronization of processors due to the fact that in each block only "own" processors have access to the local RAM bank, high performance and effectiveness, practically proven on MPP -machines with a large number of processors (several thousand) [8].

The hypercube is one of the most widespread topologies, in particular for MPP systems, and has well-described characteristics and information on its application for various tasks. This topology is a special case of the grid structure, when there are only two processors for each dimension of the grid (that is, the hypercube contains 2N processors with dimension N) [9].

The hypercube topology is quite widespread in practice when combining parallel processors. A line connecting two nodes defines a one-dimensional hypercube. A square formed by four nodes is a two-dimensional hypercube, and a cube with eight nodes is a three-dimensional hypercube, etc. Since the system consists of several processor elements with local memory, the time spent on data transfer must be taken into account when performing the task. The following diagram shows the data transfer algorithm between nodes during DFT execution for 4 signals.

Based on the algorithm (Fig. 1), we will set the time functions of sequential processing and parallel processing

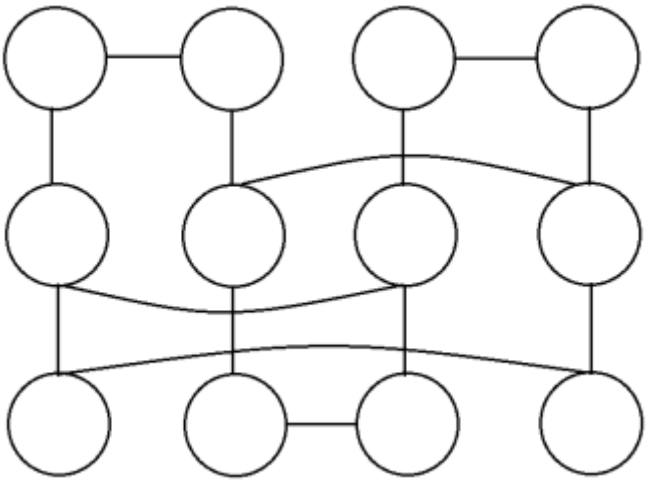

Figure 1. Interaction diagram of processors for the MPP system with parameters N and n=4

Let's apply the acceleration and efficiency formulas to obtain the analytical isoefficiency formula.

$$T_{1n} = n + k_n n \tag{9}$$

$$T_{pn} = n + k_n \frac{n}{N} \tag{10}$$

$$E_n = \frac{T_1}{NT_p} = \frac{1+k_n}{N+k_n} \quad (11)$$

where N is the number of processors, n is the dimension of the problem, kn is the dimension factor. The dimensionality factor can be calculated using the following formula:

$$k_n = 2k_{\frac{n}{2}} \quad (12)$$

$$k_2 = 1 \quad (13)$$

For the example shown in Figure 1, where N=4 and n=4, the value of parallel processing time will be:

$$T_{pn} = n + k\frac{n}{N} \quad (14)$$

Several different systems were used to obtain heterogeneous system results, listed in (table 1). The systems presented use different GPUs and CPUs, which adds complexity to the analytical approach to establishing isoefficiency.

Table 1.

№	Процесор	Графічний процесор
1	AMD Ryzen 9 3900X	RTX 2060
2	AMD Ryzen 5 2400G	GTX 1060-3GB
3	Intel Core i5-7200U	Geforce 940MX
4	Intel Core i5-9600KF	GTX 1080

First, it is necessary to set the execution time of tasks with the same dimension on the processor and graphics accelerator (Fig. 2). Based on the received data, a model will be developed that will allow you to distribute the load between processors in order to complete the task as quickly as possible.

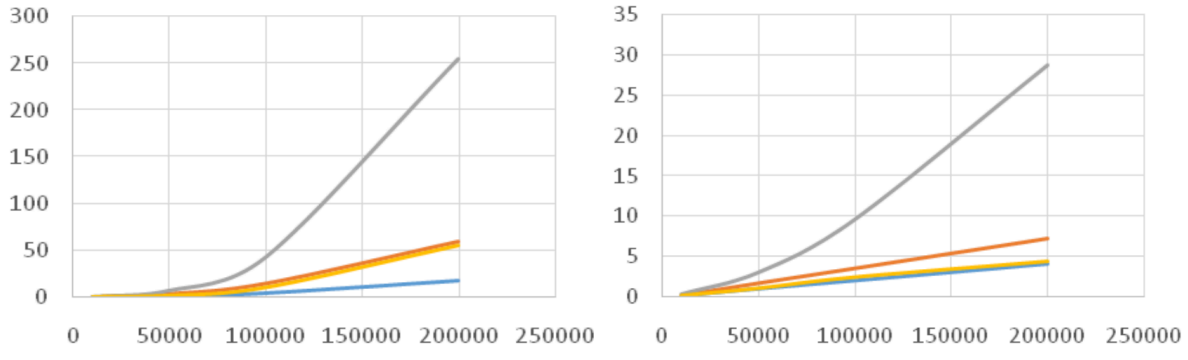

Figure 2. Task execution time on CPU (left) and GPU (right) [1]

The first thing that can be established from the graphs is the nonlinear complexity of the problem-solving algorithm. This necessitates the use of machine learning algorithms for the approximation of a nonlinear function. You can also see the difference between how the time increment changes with the change in the dimension of the problem. Before applying machine learning to distribute tasks between the processor and the graphics card, it is necessary to establish the data transfer time between the general memory of the system and the memory on the GPU.

Using an approach based on polynomial regression, let's establish the distribution of the problem's dimension on the CPU and GPU. Based on the time metrics of task execution on systems involving both processors, it is possible to empirically establish the efficiency of the system. Consider the time it takes to send data from the GPU memory to the shared memory (Fig. 3).

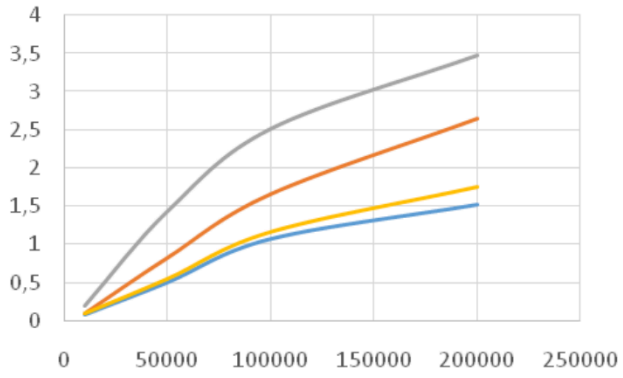

Figure 3. Time spent on sending data from GPU to shared memory [1]

Based on the diagram of the obtained time data (Fig. 4), it is possible to establish the efficiency of heterogeneous systems and establish the necessary dimension of the problem to obtain the same efficiency on another system.

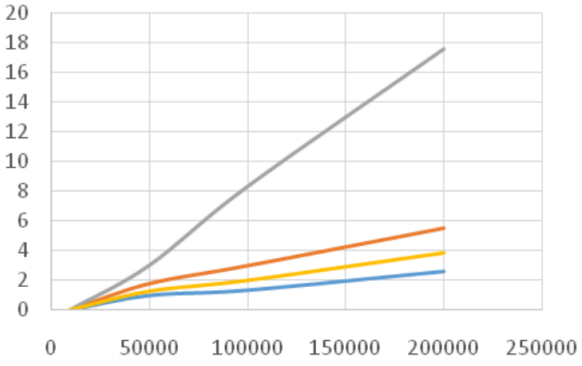

Figure 4. Task execution time on a heterogeneous system [1]

The efficiency of the system can be established based on the time of execution of the task simultaneously on the processor and on the video accelerator (Table 2).

The proposed approach below uses the distribution of the problem dimension across processors to establish the overall performance of a heterogeneous system in terms of speedup relative to the fastest processor of the two presented.

Conclusions. In the course of the research, the main result can be considered that heterogeneous CPU-GPU systems can be used for iso-efficient computing. The main advantage of this approach is predictability when building systems oriented to a specific task, within the framework of this work, such a task is the discrete Fourier transform. Based on the proposed approach, it is possible to scale the system due to accelerators based on a different architecture and develop isoefficient algorithms.

The simulation results show the nature of performing the calculation of the problem of nonlinear complexity. The use of machine learning methods, namely polynomial regression, allows you to create an algorithm for dividing the task between CPU and GPU while preserving the acceleration factor. From the results, it can also be concluded that the system with the most powerful processor shows the best results for n = 100000, while the results for n = 50000, 200000 are more balanced.

Table 2.

№	Система	Графічний процесор	n = 50000	n = 100000	n = 200000
1	AMD Ryzen 9 3900X	RTX 2060	1.004133	1.419191	1.446006
2	AMD Ryzen 5 2400G	GTX 1060-3GB	1.011028	1.360679	1.402923
3	Intel Core i5-7200U	Geforce 940MX	0.931325	1.228246	1.63615
4	Intel Core i5-9600KF	GTX 1080	1.014851	1.245034	1.465462

References

1. V. Rusinov, O. Cherevatenko, L. Pustovit, O. Pustovit, Method of Development of Isoefficient Heterogeneous System Using Machine Learning for the Problem of Discrete Transformation of Fourier, Herald of Khmelnytskyi National University, 297.3 (2021), 19–24
2. Hwang K. Scalability and programmability of massively parallel processor. Parallel Processing: CONPAR 94-VAPP VI. Springer, Berlin, Heidelberg, 1994. P. 1–4.
3. Grama A. Y., Gupta A., Kumar V. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel & Distributed Technology: Systems & Applications. 1993. Vol. 1. Issue 3. P. 12–21.
4. Drozdowski M., Singh G., Marszalkowski J. M. Isoefficiency Maps for Divisible Computations in Hierarchical Memory Systems. PPAM (1). 2019. P. 224–234.

5. Ostertagová E. Modelling using polynomial regression. Procedia Engineering. 2012. Vol. 48. P. 500–506.
6. Bracewell R. N., Bracewell R. N. The Fourier transform and its applications. New York: McGraw-Hill, 1986. Vol. 31999. P. 267–272.
7. Harish P., Narayanan P. J. Accelerating large graph algorithms on the GPU using CUDA. International conference on high-performance computing. Springer, Berlin, Heidelberg, 2007. P. 197–208.
8. Hey T., Scott C., Surridge M. Simulation and modelling applications on mpp systems. Massively Parallel Processing Applications and Development. Elsevier, 1994. P. 15–21.
9. Yongchang J. et al. A scalability metric for algorithm-machine on NOW and MPP. Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region. IEEE, 2000. Vol. 1. P. 405–407.

Authors

Volodymyr Rusinov – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Oleksii Cherevatenko – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Leonid Pustovit – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Oleksandr Pustovit – PhD student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Artem Volokyta – associate professor, PHd, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.