Mykyta Melenchukov, Artem Volokyta, Olga Rusanova

METHOD FOR CALCULATING GAUSSIAN FUNCTIONS TO SOLVE THE PROBLEM OF IMAGE BLUR ON A HETEROGENEOUS SYSTEM

The article examines the Gaussian image blurring method using heterogeneous system.

Keywords: Gaussian function, heterogeneous system, CPU, GPU.

Relevance of the research topic. Heterogeneus computations is underdeveloped currently and there are many areas where they can be efficiently used [1]. At the same time, there are many problems that are solved by algorithms, certain parts of which are better performed on a CPU or GPU[2, 3]. In addition, on not very powerful systems, simple parallelization of calculations on the CPU and GPU can significantly reduce the calculation time[4]. Performing calculations on heterogeneous systems allows you to solve these problems

Target setting. Simultaneous execution of calculations on GPU and CPU helps to overcome the disadvantages of calculation on CPU and GPU for different algorithms by optimizing the execution process and taking advantage of the strengths of both.

Actual scientific researches and issues analysis. There are many scientific studies that describe approaches to the effective use of heterogeneous systems for solving problems in various directions. They differ in approaches to the distribution of the share of calculations between the CPU and GPU, optimization methods of calculations of the algorithm parts that are not adapted to calculations on the CPU or GPU.

Uninvestigated parts of general matters defining. Heterogeneous computing is a modern topic that is rapidly developing now and has many directions for research. This article examines the effectiveness of a heterogeneous system depending on the volume of input data, its distribution between the CPU and GPU, and the computational complexity of the algorithm.

The research objective. The purpose of this work is to investigate the optimization of a heterogeneous system for the problem of Gaussian image blurring, its dependence on the volume of input data, and the complexity of blurring.

The statement of basic materials. This article considers the solution of the Gaussian image blurring problem [5] using a heterogeneous system. In this algorithm, image blurring is achieved by calculating the color of each pixel of the resulting image from the color values of its surrounding pixels (Figure 1).

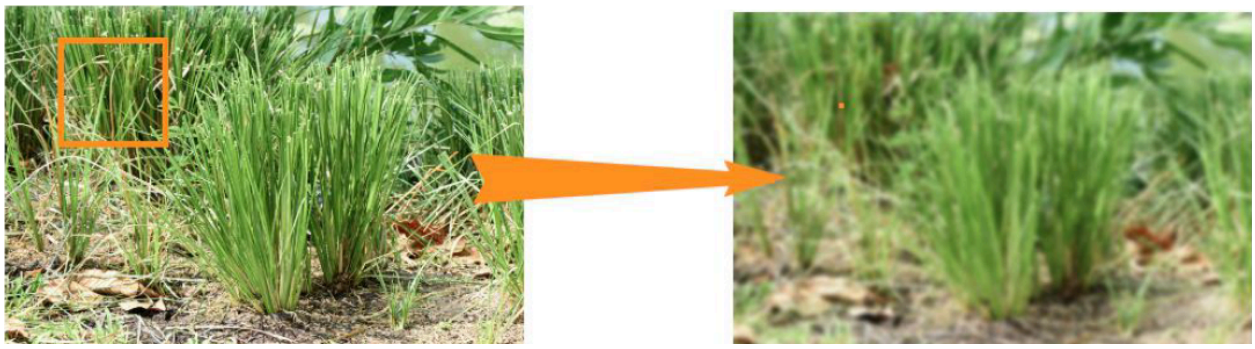

Fig. 1. The principle of operation of the Gaussian image blurring algorithm

The algorithm starts by creating matrix operator - a filter that will be applied to each image pixel. The values of the cells of this matrix are calculated by the formula (Equation 1).

$$val(i,\ j) = e^{-\frac{i^2+j^2}{2\sigma^2}} \quad (1)$$

Where σ - blur uniformity coefficient. In the next step, the matrix is normalized. For example, for a 3x3 matrix and σ = 2.411, there will be such a result (Table 1):

0.104745	0.114153	0.104745
0.114153	0.124407	0.114153
0.104745	0.114153	0.104745

Tab. 1. Example of matrix operator

The color of each pixel is encoded by RGB values - 3 numbers that represent red, green and blue color levels. For each pixel of image should be counted new value of RGB color (R_{new}, G_{new}, B_{new}) to perform blur operation (Equation 2).

$$R_{new} = \sum k_i * R_i + 0.124407 * R_{old}$$

$$G_{new} = \sum k_i * G_i + 0.124407 * G_{old}$$

$$B_{new} = \sum k_i * B_i + 0.124407 * B_{old} \quad (2)$$

Where k_i is a coefficient from matrix operator. R_{old}, G_{old}, B_{old} – previous color of pixel, R_i, G_i, B_i – colors of neighboring pixels.

An image of the size width x height is submitted to the input of the algorithm. It is processed in the form of a matrix. The total number of pixels in the image was used to compare the results on the volume of input data.

Three approaches to the solution that were chosen:

- Multi-threaded on CPU using OpenMP
- Multi-threaded on GPU using CUDA
- Combination of multi-threaded solutions on CPU and GPU.

To combine the CPU and GPU, image should be divided into columns - 75% of the columns are computed by the GPU, 25% by the CPU. The calculation of the transformation matrix was performed on the CPU in all cases.

Solutions were tested on images with resolutions of 320x320, 412x275, 600x450, 1000x563, 1200x900, 2048x1306, 4250x2833 with operator matrix sizes 3 and 5 (Table 2-3).

Image resolution	Pixel count	CPU execution time, ms	GPU execution time, ms	CPU + GPU execution time, ms
320x320	102400	14	103	17
412x275	113300	8	68	41
600x450	270000	27	69	27
1000x563	563000	41	68	26
1200x900	1080000	68	140	40
2048x1306	2674688	155	91	46
4250x2833	12040250	665	179	184
5120x2880	14745600	825	125	218

Tab. 2. Execution time for 3x3 matrix operator

Image resolution	Pixel count	CPU execution time, ms	GPU execution time, ms	CPU + GPU execution time, ms

320x320	102400	39	89	62
412x275	113300	22	67	25
600x450	270000	35	65	36
1000x563	563000	75	68	64
1200x900	1080000	210	71	57
2048x1306	2674688	390	90	110
4250x2833	12040250	1620	121	399
5120x2880	14745600	1905	146	581

Tab. 3. Execution time for 5x5 matrix operator

As the result, calculations on the CPU were the slowest, the calculations on the GPU + CPU were faster than the GPU until some critical point in the calculations (Figure 2-3), which is related to the size of the operator matrix - the larger it is, the sooner the GPU will overtake the CPU + GPU solution. With an increase in the operator matrix, the complexity of calculations grows quadratically and the CPU solution no longer has time to process its share of calculations. If it is reduced from 25% to 10%, then the separation from the GPU will come later and will not grow so fast(Figure 4).

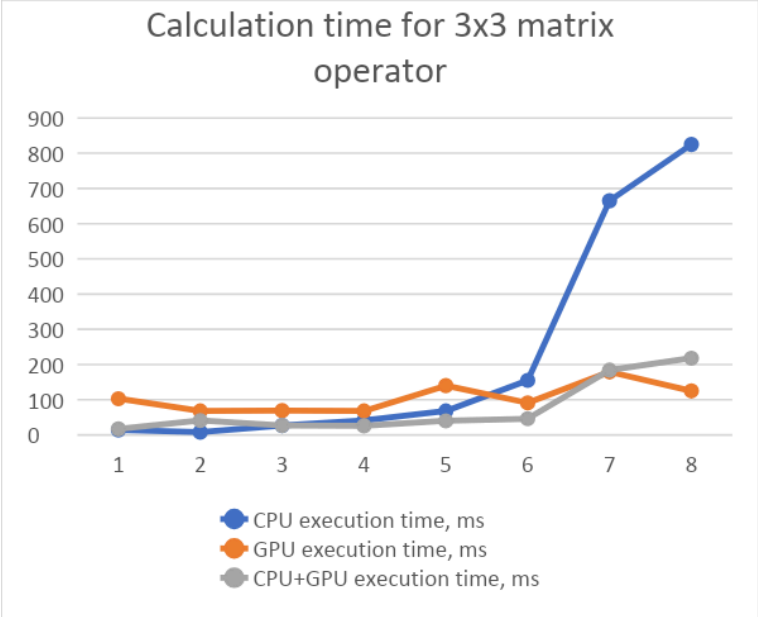

Fig. 2. Calculation time for 3x3 matrix operator

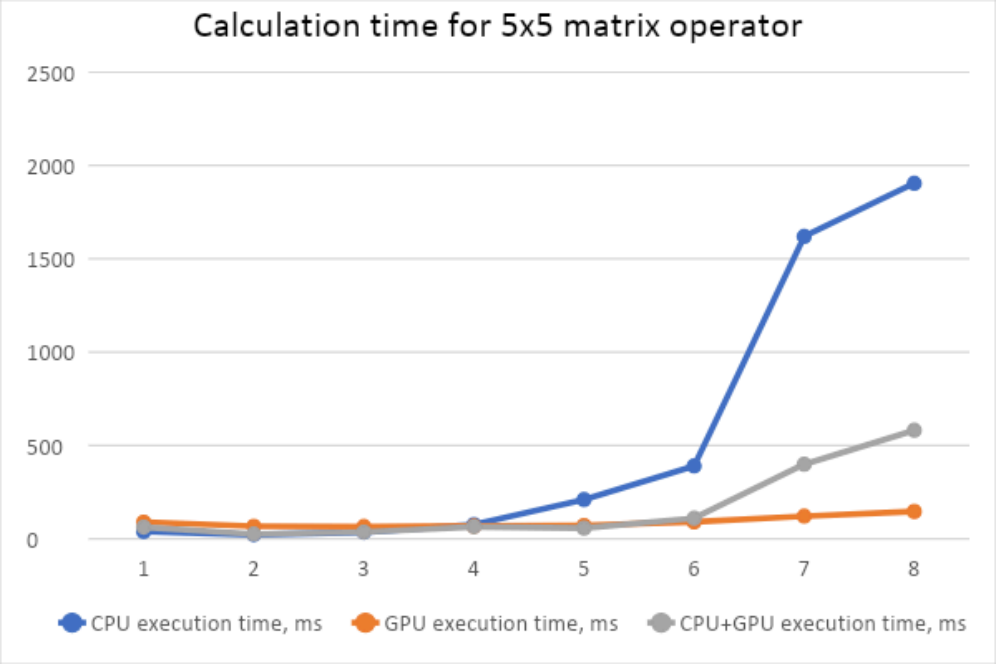

Fig. 3. Calculation time for 5x5 matrix operator

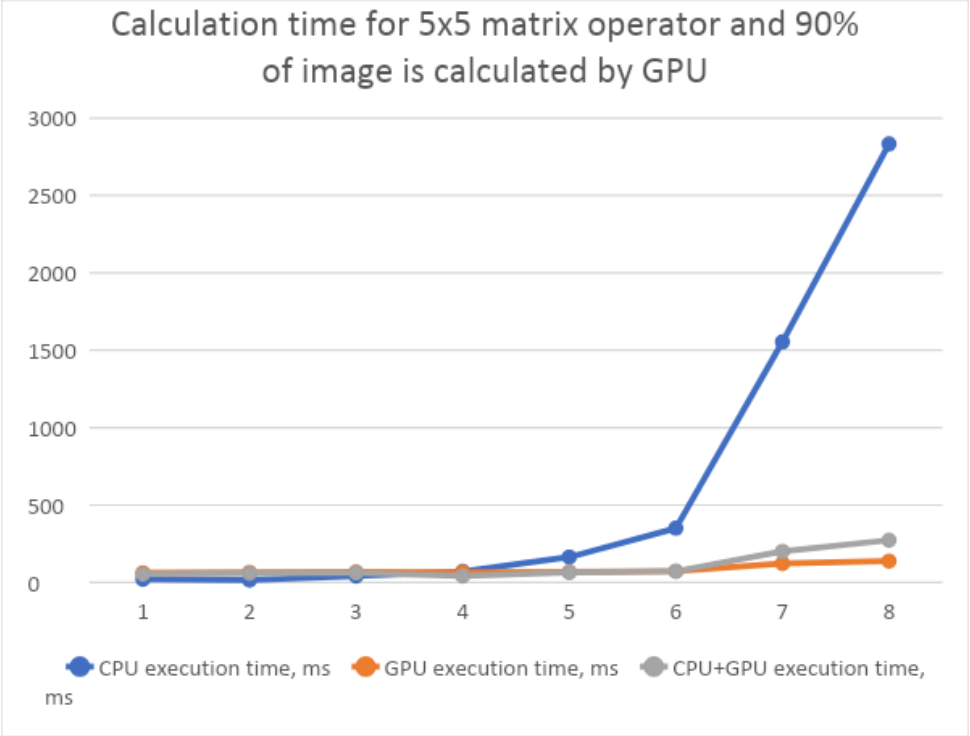

Fig. 4. Calculation time for 5x5 matrix operator and 90% of image is calculated by GPU

Conclusion. This article examines the relationship between the efficiency of using a heterogeneous system and the size of the input data, the complexity of the algorithm, the different sizes of the image portions assigned to the calculation of the GPU and the CPU.

The GPU does not highly dependent on the image size - with an increase of input data by 4.5 times (with image dimensions from 2048x1306 to 4250x2833), the computation time increased by 1.3 times.

The CPU is very dependent on the size of the input data - on the same interval, the computation time increased by 4.15 times.

The CPU+GPU solution occupies an intermediate stage and achieves better results with the correct combination of the proportions of calculations parts and their difficultness (calculated through the size of the matrix operator).

References

1. Lusher, David J., Satya P. Jammy, and Neil D. Sandham. "OpenSBLI: Automated code-generation for heterogeneous computing architectures applied to compressible fluid dynamics on structured grids." Computer Physics Communications 267 (2021).
2. Tang, Xiaoyong, and Zhuojun Fu. "CPU–GPU utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems." IEEE Access 8 (2020).
3. Benatia, Akrem, et al. "Sparse matrix partitioning for optimizing SpMV on CPU-GPU heterogeneous platforms." The International Journal of High Performance Computing Applications 34.1 (2020).
4. Bateni, Soroush, et al. "Co-optimizing performance and memory footprint via integrated cpu/gpu memory management, an implementation on autonomous driving platform." 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2020.
5. Bozkurt, Ferhat, Mete Yaganoglu, and Faruk Baturalp Günay. "Effective Gaussian blurring process on graphics processing unit with CUDA." International Journal of Machine Learning and Computing 5.1 (2015).

Authorus

Melenchukov Mykyta – student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.
E-mail: melenchukov.nikita@gmail.com

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

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