Petro Syvenkyi, Volodymyr Yarkun*
student, Ukrainian National Forestry University, Lviv, Ukraine.
p.syvenkyy@nltu.lviv.ua.
*senior teacher of Software Engineering Department, Ukrainian National Forestry University, Lviv, Ukraine yarkun@nltu.edu.ua, https://orcid.org/0000-0002-4876-1111

Application for code editing with GPGPU pipelines

Abstract. This paper presents the advantages of using GPGPU pipelines for text processing and text drawing. Problems that arise when using GPGPU pipelines for text processing and text drawing are described. For text processing and text drawing, a code editor has been developed that can communicate with the GPU and use GPGPU pipelines. The main steps to achieve these goals are outlined.

Keywords – CUDA, text processing, GPGPU pipeline, code editor.

Code editors are programs that help programmers edit code more easily and efficiently. They offer various features and tools to improve the software development process. However, most code editors rely only on CPU power for text and graphics processing, which can cause system overload and performance issues. A possible solution is to develop a code editor that uses GPGPU pipelines to optimize text processing and text drawing with the graphics card. This is an urgent problem because it can increase the speed, accuracy and concentration of programmers.

Text manipulation and text drawing are two common tasks in code editors, but they can be computationally intensive and slow down the application. To optimize these tasks, you can use GPGPU pipelines [1], which are parallel computing methods that use a graphics card (GPU) to perform general-purpose calculations. GPGPU pipelines can speed up text processing and drawing by taking advantage of GPUs' high parallelism, memory bandwidth, and specialized hardware.

To use GPGPU pipelines for text processing and text drawing, one needs to design a code editor that can communicate with the GPU and use its resources. This involves several steps, such as:

* науковий керівник

Implementing text processing algorithms on the GPU using CUDA or OpenCL, which are programming languages that enable general-purpose computing on GPUs. Text processing algorithms can include tasks such as tokenization, parsing, syntax highlighting, auto-completion, etc.

Implementing text drawing algorithms on the GPU using OpenGL[2] or DirectX, which are graphics APIs that enable rendering of 2D and 3D graphics on GPUs. Text drawing algorithms can include tasks such as font rendering, text layout, line wrapping, scrolling, etc.

Developed a software with a user interface that displays on the screen the text of the program code, processed and drawn by the GPU. This is implemented using graphical interfaces such as Qt or GTK+, which provide widgets and tools for building graphical user interfaces.

In the created program, in order to open the file, you need to specify the location and name of the program startup file in the terminal and specify the location and name of the text file that needs to be opened Fig.1. This can be easily done by opening the folder with the file in the file manager and dragging the desired file to the terminal with the left mouse button.

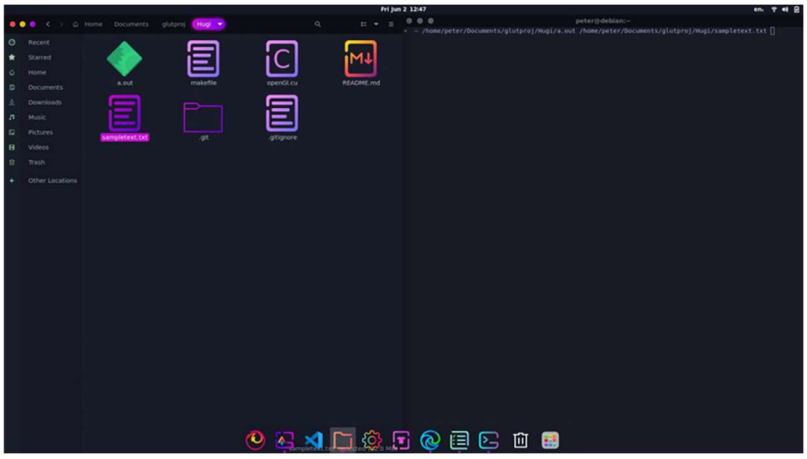

Fig. 1. Opening the file.

When starting the application, the content of the file Fig. 2 will be displayed immediately. Then you can make changes to the text of the program code. By pressing the key combination "Ctrl+S" the changes will be saved to the file.

* науковий керівник

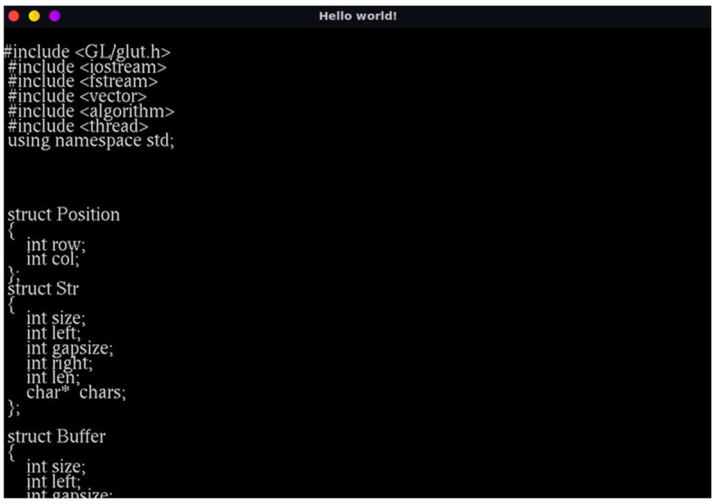

Fig. 2. Displaying the software code in the application.

The use of GPGPU pipelines for text processing and text drawing improved the performance and response speed of the code editor by reducing CPU load and using GPU power. In addition, we have improved the quality and functionality of the code editor by enabling more complex and improving existing text processing and text drawing functions. However, using GPGPU pipelines for text processing and text drawing also creates compatibility and security issues due to dependencies on GPU drivers and libraries. Also, using the developed software requires additional hardware and software resources, such as a dedicated graphics processor and GPGPU structure.

Conclusion. Code editors are essential software applications for programmers, but they often suffer from performance issues due to their heavy reliance on CPU power. A potential solution is to develop a code editor that uses GPGPU pipelines. The software is implemented taking into account the optimal use of GPGPU resources, including the selection of optimal algorithms and approaches to improve performance. This can improve the efficiency and quality of code editing, as well as the concentration and accuracy of programmers.

References

[1] Sanders, J., E. Kandrot (2010): CUDA by Example: An Introduction to General-Purpose GPU Programming.

[2] Marcus D. Hanwell, Christopher J. Harris, Alessandro Genova, Jonathan Schwartz, Yi Jiang, Robert Hovden (2019): Tomviz: Open Source Platform Connecting Image Processing Pipelines to GPU Accelerated 3D Visualization.

* науковий керівник