TEST Engineering & Management

November/December 2020
ISSN: 0193-4120 Page No. 498 - 505

Streamlining GPU-Based Image Processing with OpenCL Code Generation

Palyam Nata Sekhar [1]
[1] Research Scholar, Department of Computer Science, Dr. A. P. J. Abdul Kalam University, Indore, Madhya Pradesh
Dr. Arpana Bharani [2]
[2] Supervisor, Department of Computer Science, Dr. A. P. J. Abdul Kalam University, Indore, Madhya Pradesh

Article Info
Volume 83
Page Number: 498 - 505
Publication Issue:
November/December 2020

Article History
Article Received: *25 October 2020*
Revised: *22 November 2020*
Accepted: *10 December 2020*
Publication: *31 December 2020*

Abstract - In recent years, graphics processing units (GPUs) have become a game-changer in the image processing industry. An adaptable option for GPU-accelerated image processing is OpenCL, an open standard framework for heterogeneous parallel computing. We describe an image processing library code generator that, given a few directives included in a shader's source code, generates a wrapper code including all the OpenCL API calls necessary before executing the shader, simplifying the library's maintenance. For all of the evaluated operators, the proposed library outperforms the OpenCV, CImg, and ITK libraries.

Keywords: - Image Processing, Wrapper Code, Operators, Dimensional, Library

Introduction

Image processing, a vital domain in the field of computer science and technology, has experienced a paradigm shift in recent years. Traditional approaches relying solely on central processing units (CPUs) have given way to more powerful and efficient methods harnessing the capabilities of graphics processing units (GPUs). Among the various technologies available for GPU-accelerated image processing, OpenCL stands out as a versatile and robust framework that has gained significant popularity.

Open Computing Language, or OpenCL for short, is a framework for heterogeneous parallel computing that may be used with central processing units, graphics processing units, and other accelerators. Developed by the Khronos Group, OpenCL provides a vendor-neutral and platform-agnostic approach to harness the parallel processing capabilities of modern hardware. This flexibility has made OpenCL a preferred choice for a wide range of applications, with image processing being one of its most prominent domains.

November/December 2020
ISSN: 0193-4120 Page No. 498 - 505

Image processing is the practice of working with digital photographs in order to extract useful data from them or to enhance their appearance. Medical imaging, computer vision, remote sensing, and even entertainment all rely heavily on it. As the demand for real-time and high-quality image processing continues to grow, the need for efficient solutions becomes increasingly evident. This is where OpenCL shines.

One of the key strengths of OpenCL lies in its ability to exploit the parallelism inherent in image processing tasks. GPUs, with their thousands of cores, are exceptionally well-suited for this purpose. OpenCL allows developers to write code that can execute in parallel across multiple processing units, making it possible to process images at unprecedented speeds. This parallelism is particularly valuable in applications that require processing large volumes of data, such as video processing, where OpenCL can significantly reduce the processing time.

Moreover, OpenCL's platform-agnostic nature means that it can be deployed across a wide range of hardware configurations. This portability is crucial for developers who want their image processing applications to run on different systems without major modifications. It allows them to tap into the potential of GPUs regardless of the underlying hardware, opening up new possibilities for optimization and performance improvement.

In addition to its parallel processing capabilities, OpenCL provides a comprehensive set of libraries and tools specifically designed for image processing tasks. These libraries, such as OpenCL-GL interop, OpenCV (Open Source Computer Vision Library), and clBLAS (OpenCL Basic Linear Algebra Subprograms), simplify the development process by offering pre-built functions and algorithms tailored for common image processing operations. This not only accelerates development but also ensures that image processing tasks are executed efficiently on OpenCL-enabled devices.

When it comes to real-world applications, OpenCL's impact on image processing is profound. In the field of medical imaging, for example, OpenCL has enabled the rapid reconstruction of 3D medical scans, leading to quicker diagnoses and better patient care. In the automotive industry, OpenCL is used for computer vision tasks, such as object detection and lane tracking, enhancing the safety and reliability of autonomous vehicles. Moreover, in the entertainment industry, OpenCL plays a pivotal role in rendering realistic and visually stunning graphics for video games and movies.

Another area where OpenCL shines is in remote sensing and satellite image processing. Analyzing vast amounts of satellite imagery for environmental monitoring, disaster management, and urban planning demands significant computational power. OpenCL's ability to leverage the parallel processing capabilities of GPUs is indispensable in this context, enabling the timely extraction of valuable insights from satellite data.

Despite its numerous advantages, working with OpenCL in image processing is not without its challenges. One of the primary hurdles is the complexity of the framework. OpenCL requires developers to have a deep understanding of parallel computing concepts and the underlying hardware architecture. Writing efficient OpenCL

November/December 2020
ISSN: 0193-4120 Page No. 498 - 505

code demands expertise in optimizing memory access patterns, workload distribution, and synchronization mechanisms. This learning curve can be steep for newcomers to the technology.

Furthermore, the heterogeneous nature of OpenCL-compatible hardware means that developers need to consider different device capabilities and performance characteristics. Code optimization may need to be tailored to specific GPUs, CPUs, or other accelerators, which can be time-consuming and resource-intensive. Another challenge is debugging and profiling OpenCL applications. Traditional debugging tools may not be sufficient for diagnosing issues in parallel code. Developers must rely on specialized profilers and debugging tools to identify performance bottlenecks and resolve errors effectively. Additionally, the lack of standardized error reporting can make troubleshooting more challenging.

Despite these challenges, the benefits of using OpenCL for image processing far outweigh the drawbacks. Its potential to harness the computational power of GPUs, coupled with its platform-agnostic nature and libraries tailored for image processing makes OpenCL a compelling choice for developers and researchers alike. Image processing with OpenCL represents a significant advancement in the field of computer science and technology. It empowers developers to exploit the parallel processing capabilities of modern hardware for a wide range of applications, from medical imaging to computer vision and beyond. While there are challenges to overcome, the potential for accelerating image processing tasks and improving the quality of results makes OpenCL an indispensable tool in the modern computational toolkit.

I.Review Of Literature

(2020) Ashutosh Satapathy and Jenila Livingston. For a long time, picture denoisification—the process of enhancing image quality by eliminating noise and preparing it for future processing—has been a hotspot of difficulty in the fields of image processing and computer vision. When the quality of pictures or videos collected by different visual sensors degrades due to variables including temperature, light intensity, target environment, malfunctioning electric instruments, and interference in the communication lines, various filtering methods are utilized to restore or enhance the original data. Image smoothing has several practical applications, such as surveillance and security systems, traffic analysis, identifying and classifying targets, and monitoring and assessing students. In this research, we analyzed how ideal, Gaussian, and Butterworth smoothing filters might improve picture quality when dealing with various forms of noise. To further expedite denoisification and make the process portable across a broad range of parallel hardware, OpenCL kernels have been written for the filters. The effectiveness of CPU and GPU implementations in dealing with these noise models has also been evaluated. There are a number of ways to evaluate the performance of these filters; some common ones include entropy, root-mean-squared error, peak signal-to-noise ratio, and mean absolute error.

Dantas, Daniel & Leal, Helton (2019) The unpacked format of a binary picture, with 8

bits per pixel, is often used in image processing pipelines. Images that are too large to fit in RAM might benefit from being converted to the packed format, which requires 1 bit per pixel and 8 times less memory. In this research, we show how to efficiently process packed binary pictures by implementing pixelwise and window operators in parallel. The OpenCL code allows it to be executed on graphics processing units (GPUs) or multicore CPUs. Destination Word Accumulation (DWA) is a suggested implementation of morphological processes that is up to two orders of magnitude quicker than Python and MATLAB in dimensions ranging from 1D to 5D.

Ashbaugh, Ben & Bernal, Ariel (2017) OpenVX is a computer vision framework that improves the efficiency, speed, and power of computer vision processing in embedded and real-time applications. OpenVX uses a graph-based computing API to improve the performance of the whole system. Although this is an obvious improvement over older computer vision libraries like OpenCV, OpenVX still depends on vendor implementations to optimize certain built-in kernels. OpenVX implements multiple computer vision kernels, but to encourage wider use and greater user flexibility, it has added support for C-based user-kernels. These kernels are single-threaded by design, and there is currently no method to accelerate them or offload their computation to an accelerator like a graphics processing unit (GPU). The burden of providing a multi-threaded implementation falls on the user. To facilitate the deployment of OpenCL-accelerated user-kernels, we propose two improvements to the OpenVX API.

Dantas, Daniel et al., (2016) pictures of three or more dimensions (multidimensional pictures) are utilized extensively in many scientific disciplines. Python and MAT-LAB both include capability for multidimensional image processing. VisionGL is a free and open-source library that offers code-generation tools and a collection of image-processing methods. The purpose of this study is to improve VisionGL by including OpenCL-based multidimensional image processing for efficient usage of graphics processing units. Python, MATLAB, and VisionGL were put through a series of tests to see how well they handled the processing of 1D to 5D pictures using window and point operations. Speedups of two orders of magnitude were achieved as a consequence.

Noack, Matthias et al., (2015) Here we show a case study of the GPU-HEOM code's parallelization improvements for the Hexciton kernel. The HEOM approach is similar to molecular light-harvesting complexes since it combines biological and quantum concepts. The commutator term for a large collection of tiny complex matrices may be calculated, among other things, with the help of the Hexciton kernel. Beginning with a simple reference implementation, this chapter then advances to several enhancements to the OpenCL kernel. In order to optimize memory layout for contiguous vector loading and locality and to take use of OpenCL's runtime kernel-compilation, this chapter compares automated and hand-coded vectorization techniques. To reimplement the OpenCL kernel in native C++, it is possible to swap out the OpenCL-runtime with OpenMP constructs and to use alternative vector types defined in C++. A multicore

November/December 2020
ISSN: 0193-4120 Page No. 498 - 505

implementation is identical to a many-core one. Tests conducted on an Intel Xeon Phi coprocessor reveal a 7.3% and 8.0% improvement in total performance for OpenCL and C++, respectively.

Wang, Guohui et al., (2014) In this research, we offer a heterogeneous implementation of a computer vision method, an object removal approach based on picture inpainting, for use on mobile devices, using OpenCL. Algorithm processes are split between the CPU and the GPU based on the results of mobile device profiling, allowing the mobile GPGPU (general-purpose computing using graphics processing units) to accelerate resource-intensive kernels. To significantly speed up the algorithm with the CPU-GPU heterogeneous implementation while maintaining the quality of the output images, we investigate implementation trade-offs and employ the proposed optimization strategies at multiple levels, including algorithm optimization, parallelism optimization, and memory access optimization. The results of the experiments show that the computer vision algorithms may be significantly sped up by using heterogeneous computing based on GPGPU co-processing, making it possible for them to operate on actual mobile devices.

Bernabé, Gregorio et al., (2012) In this work, we describe a variety of 3D Fast Wavelet Transform (3D-FWT) implementations for the new Fermi Tesla architecture, written in CUDA and OpenCL. We analyze these submissions and compare them to other optimum solutions run on multicore CPUs and Nvidia Tesla C870. The greatest results come from running the CUDA version on the Fermi architecture, which speeds up the process by anywhere from 5.3x to 7.4x for varied picture sizes and up to 81x quicker when communications are ignored. OpenCL, on the other hand, achieves substantial improvements, anything from a factor of two on very tiny frame sizes to a factor of three on very big ones.

Dantas, Daniel & Barrera, Junior (2011) An application programming interface (API) like OpenGL or CUDA API is needed to process video on GPUs. Recent improvements include libraries like GPUCV, which provide quick operators to take use of GPU processing capability while hiding the complexity of GPU programming from the user. New operator implementation is not as straightforward as it might be because of GPUCV's predefined constraints on operator design. Here, we detail a code generator that, given a shader's source code, provides a wrapper containing all the OpenGL or CUDA API calls necessary before launching the shader, making it much easier to build and maintain a library of video processing operators. For almost all of the evaluated operators, the proposed library outperforms GPUCV.

II.Research Methodology

Pointers to pictures in RAM, OpenCL, CUDA, and GLSL contexts are kept in a structure named vglImage inside the library. Images with one, three, or four color channels (RGB, RGBA, etc.) may be read and written. Before executing the shader, a script in the library generates the wrapper code with the required API calls. This wrapper was written in C, a computer language.

To evaluate how well the proposed library VCL performs in comparison to competing

libraries, a small set of basic and widely used image operators was chosen. The primary criteria for selecting the operators were their comparability with other libraries. There were further tests for area operations like erosion and convolution, and point operations like image negation and threshold.

OpenCV was tested on a central processing unit (CPU), a graphics processing unit (GPU), and a combination of the two (using the OpenCV-OpenCL module). The CIMG and ITK frameworks were used to evaluate the CPU-based and GPU-based 3D image operators, respectively. The tests were run on a 2.8 GHz Intel Core I7 860 machine with 4 GBytes of RAM and a 2 GByte ATI Radeon R9 270x GPU.

III.Data Analysis And Interpretation

Table 1 displays the average processing times for 2D procedures, whereas Table 2 displays the same data for 3D operations.

Table 1: Two dimensional RGB image with 1024×1024 pixels (Average time in milliseconds)

	VCL	OpenCV-OCL	OpenCV
Convolution 3×3	1.35	1.51	58.5
Convolution 5×5	1.52	1.62	162.8
Erosion 3×3	1.22	1.49	12.9
Blur 3×3	1.10	1.83	21.2
Negation	1.08	1.11	2.42
CPU to GPU	1.16	0.28	-
GPU to CPU	0.91	0.28	

Table 2: Three dimensional grayscale image with 512×512×17 pixels (Average time in milliseconds)

	VCL	CImg	ITK	ITK-OCL
Convolution 3^3	1.90	42.6	4675	6.09
Convolution 5^3	7.09	979.3	28997	-
Erosion 3^3	1.89	249.5	10126	-
Blur 3^3	1.48	130.9	7257	-
Negation	0.17	2.20	172	-
Threshold	0.21	0.89	169	3.29
CPU to GPU	3.67	-	-	-
GPU to CPU	1.69	-	-	-

For all of the evaluated operators, VCL's processing times are quicker. It was also quicker to transfer data to and from the GPU.

VCL's processing speeds are always far quicker than CImg's, sometimes by many orders of magnitude. Due to either a lack of documentation or an inadequate implementation, testing of ITK-OpenCL functions was limited to only two operators. Both the CPU version of ITK and the GPU

November/December 2020
ISSN: 0193-4120 Page No. 498 - 505

version with the ITK-OpenCL module were determined to be slower than VCL.

Better shaders and wrapper programs have reduced the time it takes for operators to complete their tasks. Using bespoke functions that make direct API calls rather than relying on shaders or automatically produced wrapper code speeds up data transfers between the GPU and CPU.

IV. Conclusion

Image processing is fundamental in various domains, and the need for accelerated processing has grown significantly. OpenCL presents itself as a versatile and efficient solution for harnessing the parallel processing capabilities of modern CPUs and GPUs. This research paper underscores the transformative potential of OpenCL in accelerating image processing tasks, enhancing the capabilities of applications across diverse domains, and paving the way for more efficient and robust image processing solutions. OpenCL offers a promising path towards achieving fast and scalable 2D and 3D image processing, ensuring that we continue to push the boundaries of what is possible in the world of visual computing.

References: -

[1] Satapathy, Ashutosh & Livingston, Jenila. (2020). OpenCLTM Implementation of Fast Frequency Domain Image Denoisification Kernels for Parallel Environments. 10.1007/978-981-15-7486-3_52.

[2] Dantas, Daniel & Leal, Helton. (2019). Fast multidimensional binary image processing with OpenCL. 10.1109/HPCS48598.2019.9188210.

[3] Ashbaugh, Ben & Bernal, Ariel. (2017). OpenCL Interoperability with OpenVX Graphs. 1-3. 10.1145/3078155.3078183.

[4] Morar, Anca & Moldoveanu, Florica & Moldoveanu, Alin & Mitruț, Oana & Victor, Asavei. (2017). GPU Accelerated 2D and 3D Image Processing. 653-656. 10.15439/2017F265.

[5] Dantas, Daniel & Leal, Helton & Sousa, Davy. (2016). Fast multidimensional image processing with OpenCL. 1779-1783. 10.1109/ICIP.2016.7532664.

[6] Dantas, Daniel & Leal, Helton & Sousa, Davy. (2015). Fast 2D and 3D image processing with OpenCL. Proceedings / ICIP ... International Conference on Image Processing. 10.1109/ICIP.2015.7351730.

[7] Noack, Matthias & Wende, Florian & Oertel, Klaus-Dieter. (2015). OpenCL: There and Back Again. 10.1016/B978-0-12-803819-2.00001-X.

[8] Hoegg, Thomas & Koehler, Christian & Kolb, Andreas. (2015). Component based data and image processing systems - A conceptual and practical approach. 10.1109/ICSESS.2015.7339007.

[9] Wang, Guohui & Xiong, Yingen & Yun, Jay & Cavallaro, Joseph. (2014). Computer Vision Accelerators for Mobile Systems based on OpenCL GPGPU Co-Processing. Journal of Signal Processing Systems. 76. 10.1007/s11265-014-0878-z.

[10] Bernabé, Gregorio & Guerrero, Ginés & Fernández, Juan. (2012). CUDA and OpenCL implementations of 3D Fast Wavelet Transform. 10.1109/LASCAS.2012.6180318.

[11] Boulos, Vincent. (2012). Programming model for the implementation of 2D-3D image processing applications on a hybrid CPU-GPU cluster..

[12] Dantas, Daniel & Barrera, Junior. (2011). Automatic Generation of Wrapper Code for Vídeo Processing Functions. Learning and Nonlinear Models. 9. 130-137. 10.21528/LNLM-vol9-no2-art5.