International Journal of Research in Engineering, Science and Management
Volume-2, Issue-3, March-2019
www.ijresm.com | ISSN (Online): 2581-5792

Efficient Batch Processing of Satellite Imagery using Open Source Tools

Ankita Mishra[1], Bhushan Dhapodkar[2], Sheetal Meshram[3], Sneha Bagde[4], Praveen Sen[5]
[1,2,3,4]*Student, Dept. of Information Technology, St. Vincent Pallotti College of Engg. & Tech., Nagpur, India*
[5]*Asst. Professor, Dept. of Information Technology, St. Vincent Pallotti College of Engg. & Tech., Nagpur, India*

***Abstract*: This paper provides an efficient study of the implementation of parallel image processing using CUDA on NVIDIA GPU framework. By utilizing CUDA as a computational resource, it provides significant acceleration in the computations of different image processing algorithm. Specifically, this approach demonstrates the efficiency by a parallelization and optimization of the algorithm. It also provides performance comparison with or without GPU.**

***Keywords*: Image Processing, CUDA (Compute Unified Device Architecture), GPU(Graphics Processing Unit), Parallel Computing.**

1. Introduction

Nowadays, computers process a huge amount of data [1]. For real time image processing techniques, high performance is required. It is obvious for such calculations; the performance of common personal computers is insufficient. To handle this issue, parallel processing of the images is found to be the most effective approach. For this efficient and quick implementation of image processing parallelly, there are several tools and techniques available such as CUDA, GPU, PCT of MATLABTM, Open-CV AND Open-CL [2].

Parallel computing can be implemented in central processor (CPU) [3] as well as in graphical processor (GPU) [4]. Recently, graphic processing units have evolved into an extremely powerful computational resource. For example, The NVIDIA GeForce GTX 280 is built on a 65nm process, with 240 processing cores running at 602 MHz, and 1GB of GDDR3 memory at 1.1GHz running through a 512-bit memory bus. Its Peak processing power is 933 GFLOPS [5], billions of floating-point operations per second, in other words. As a comparison, the quad-core 3GHz Intel Xeon CPU operates roughly 96 GFLOPS [6]. The annual computation growth rate of GPUs is approximately up to 2.3x. In contrast to this, that of CPUs is 1.4x [6]. At the same time, GPU is becoming cheaper and cheaper.

As a result, there is strong desire to use GPUs as alternative computational platforms for acceleration of computational intensive tasks beyond the domain of graphics applications. This paper is divided into six sections. Section II describes the literature survey done for the research purpose, Section III describes implementation using GPU, Section IV describes the conclusion and future scope.

2. Literature survey

In [2], Sanjay Saxena has said the problem of processing is that it is generally time consuming. So parallel computing provides an efficient and convenient way to address the issue. There are various tools and techniques used by different researchers for implementing parallel image processing like PVM MATLAB using Mex files, Multithreading and CUDA programming with GPU. It is found that all techniques provide good speed and parallel efficiency but in multithreading technique, it needs more data to test with different performance measuring parameters, while in CUDA it needs to be tested variety of images with large dimension.

In [7], Yang has said that the graphics card that support CUDA are GeForce8-series, QUADRO and TESLA. They are specifically design to run for non-graphic purposes. There software development kit includes libraries, debugging, profiling and compiling tools. In [4], Owens J D has said that in NVIDIA GPU computing model, CPU and GPU work together in a heterogenous co-processing computing model. It provides portability, programmability and flexibility. It condensed power consumption.

In [8], Placido Salvatore has done median filtering by using GPU. They have used different CUDA fundamentals and methods for implementing median filtering and found that it is possible to get gain in response time with an access level GPU, allowing real-time image and audio filtering. Though, the bottleneck of these systems is the PCI Express bus, for devoted and straight bus throughout GPU/RAM and CPU/ GPU the response time is condensed.

In [9], S. Sharma has implemented first order edge detectors such as Roborts, Sobel and Prewitt. They perfectly utilized CUDA's huge amount of threads on GPU Ge Force GTX 860M and got significant results. They have tested 60 images consisting three different size (512 X 512, 1024 X 1024, 3072 X 3072) and found speed up around hundred times in terms of percentage.

3. Implementation

- *GPU:* GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs. The graphics processing units are extremely parallel, rapidly gaining maturity as a powerful device for computationally demanding applications. The GPU's performance and potential will be the future of computing systems [12]. A GPU is mainly designed for some particular type of applications with the following characteristics; Where Computational requirements are large: GPU must deliver an enormous amount of computing power to cover the requirements of complex real-time applications. Parallelism is significant: The graphics pipeline system architecture is appropriate for parallelism. The architectural comparison of CPU with GPU is more suitable for stream computations. They can process data elements in parallel with SIMD & MIMD capability. So a new technique called GPGPU (General Purpose computation on GPU) emerged and in recent years has become a hot research topic in not only graphics domain but also in general computations [10]. GPGPU is a combination between hardware components and software that allows the use of a traditional GPU to perform computing tasks that are extremely demanding in terms of processing power [11].

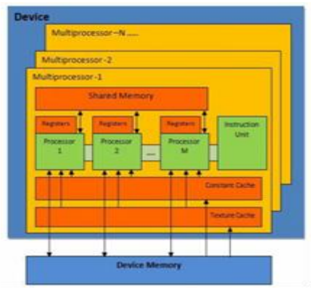


Fig. 1. Architecture of GPU

- *Architecture of GPU:* The research carried out on NVIDIA based GPU hardware using CUDA, a general purpose parallel computing architecture. The architecture of the GPU has developed in a different direction than that of the CPU. The design of the GPUs is forced by the fast growing video game industry that exerts marvelous economic pressure for the ability to perform a massive number of floating-point calculations per video frame in advanced games. The general philosophy for GPU design is to optimize for the execution of huge number of threads. Figure 1 shows the architecture of a typical GPU today. It is organized into 16 highly threaded streaming Multiprocessors (SMs). A pair of SMs forms a building block. Each SM has 8 streaming processors (SPs), for a total of 128 (16*8) SPs. Each SP has a multiply-add (MAD) unit and an additional multiply (MUL) unit. Each GPU currently comes with 4 megabytes of DRAM. These DRAMs differ from the system memory DRAMs on the motherboard in that they are essentially the frame buffer memory that is used for graphics. For graphics applications, they hold high-definition video images, and texture information for 3D rendering as in games. But for computing, they function like very high bandwidth off-chip cache, though with somewhat more latency regular cache or system memory.

Programming model of GPU: As shown in Fig. 2, the workflow of a typical scenario consists of 4 steps:

- Copy processing data
- Instruct the processing
- Execute parallel in each core
- Copy the result

Firstly, the processing data is copied from the Main Memory to the GPU Memory. From CUDA version 6.0+ a new improvement has been introduced called Unified Memory. Unified Memory hides the fact that CPU memory and GPU memory are physically separated: programming becomes easier and performance increases because of how CUDA migrates the data using asynchronous streams. Once the data is copied, the CPU instructs the process to the GPU about the computation to be off loaded. The computation can now begin. CUDA devices also offer shared memory among threads which acts as a cache which is roughly 100x faster than global memory. Lastly, the results are gathered and copied back to the Main Memory. As the reader may realize, the loading and gathering time could be critical to determine whether the parallel computing could outstand the CPU.

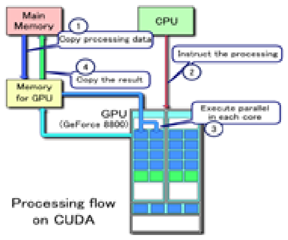


Fig. 2. Workflow of NVIDIA CUDA

Parallel computing is the simultaneous use of multiple computing resources in order to solve a computational problem. To maximize the benefits of parallel computing, a task should consist of several independent subtasks called as threads. In order to organize threads running in parallel on the GPU, CUDA organizes them into logical blocks as shown in figure 3. Each block is mapped onto a multiprocessor in the GPU.

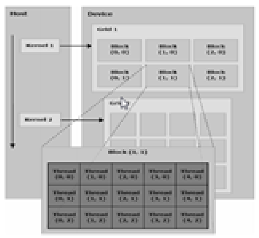


Fig. 3. Thread and block structure of CUDA

The proposed system will process different kinds of image. The main aim of our system is to increase the speed of processing the image i.e. increasing the efficiency as well as to process multiple images simultaneously.

Experimental Results: In order to relate the performance of GPU with CPU implementations, the experimental Settings are as follows:

CPU: Intel(R) Core (TM) i5-4590 at 3.30 GHz and 8GB of memory;

GPU: NVIDIA GeForce 970 GTX with 1664CUDA cores,8GB of memory, 1050MHZ clock and CUDA version of 10.1.105.

System: Windows 10 with 64-bit OS.

Images comprise millions of pixel and each pixel information is independent of its neighboring pixel. More specifically, this paper focuses on Compute Unified Device Architecture as its parallel programming platform and observes the possible gain in time which can be attained to process the single or multiple images of pixel size 10980 x 10980. The timing for reading, calculation and writing is provided in the figure. The input image is in multiple bands(RGB) and the output image is in gray band. The result shows that the execution on GPU can get a speedup as compared with the CPU.

The processed images are given below:

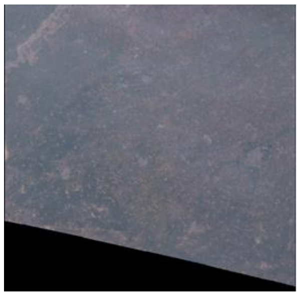


Fig. 4. Input image

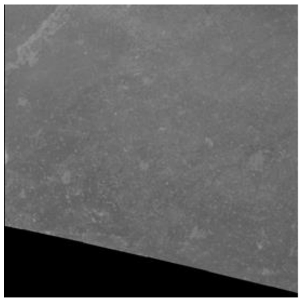
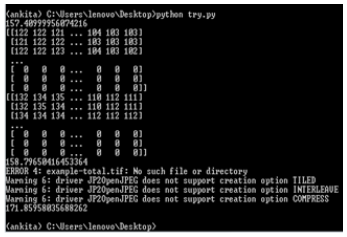


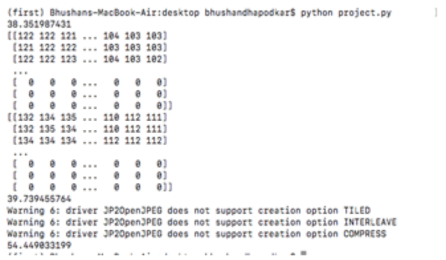
Fig. 5. Output Image



```
(ankita) C:\Users\lenovo\Desktop>python try.py
157.40999956074216
[[122 122 121 ... 104 103 103]
 [121 122 122 ... 103 103 103]
 [122 122 123 ... 104 103 102]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
[[132 134 135 ... 110 112 111]
 [132 135 134 ... 110 112 111]
 [134 134 134 ... 112 112 112]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
158.79650416453364
ERROR 4: example-total.tif: No such file or directory
Warning 6: driver JP2OpenJPEG does not support creation option TILED
Warning 6: driver JP2OpenJPEG does not support creation option INTERLEAVE
Warning 6: driver JP2OpenJPEG does not support creation option COMPRESS
171.85958035688262

(ankita) C:\Users\lenovo\Desktop>
```

Fig. 6. CPU Timing of processing Image



```
(first) Bhushans-MacBook-Air:desktop bhushandhapodkar$ python project.py
38.351987431
[[122 122 121 ... 104 103 103]
 [121 122 122 ... 103 103 103]
 [122 122 123 ... 104 103 102]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
[[132 134 135 ... 110 112 111]
 [132 135 134 ... 110 112 111]
 [134 134 134 ... 112 112 112]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
39.739455764
Warning 6: driver JP2OpenJPEG does not support creation option TILED
Warning 6: driver JP2OpenJPEG does not support creation option INTERLEAVE
Warning 6: driver JP2OpenJPEG does not support creation option COMPRESS
54.449033199
```

Fig. 7. GPU Timing of processing Image

4. Conclusion and future scope

The use of parallel computing with the GPU can significantly accelerate the implementation of program. The degree of parallelism and the acceleration is determined by the number of independent computations performed simultaneously. However, performance gain varies with the size of the images, size and shape of the used structuring element. Huge performance gain is accomplished with larger size of images with larger numbers of CUDA cores. From timing given above, we can conclude that the GPU implementation is faster than the CPU implementations. For future perspective we can implement shared memory parallel computation through an interconnection network for fastest computing.

References

[1] Korovin A S, Skirnevsky I P and Abdrashitova M A 2015 Web-Application for Real-Time Big Data Visualization of Complex Physical Experiments Proc. The 2015 Int. Siberian Conf. On Control and Communications (SIBCON) (Omsk), vol. 1, (Novosibirsk: IEEE Russia Siberia Section) pp. 1-5.

[2] Sanjay Saxena, Shiru Sharma, Neeraj Sharma, “Study of Parallel Image Processing with the Implementation of HGW Algorithm using CUDA on NVIDIA’S GPU Framework,” Proc. of the World Congress on Engineering, 2017, Vol. 1, WCE 2017, July 5-7, 2017, London, U.K.

[3] Ivy K L 1988 A shared virtual memory system for parallel computing Int. Conf. on Parallel Computing pp. 94-101.

[4] Owens J D, Houston M, Luebke D, Green S, Stone J E, and Phillips J C 2008 GPU computing Proc. of the IEEE 96(5) 879-899.

[5] NVIDIA, CUDA Programming Guide Version 2.3. NVIDIA Corporation: Santa Clara, California Intel, Quad-Core Intel® Xeon® Processor 5400 Series 2008, Intel Corporation: Santa Clara, California (2008), http://gpgpu.org, General-Purpose Computation on Graphics Hardware.

[6] Allard, J., Raffin, B.: A shader-based parallel rendering framework. In: Visualization, 2005, VIS 2005, pp. 127–134. IEEE, Los Alamitos (2005).

[7] Yang, Z., Zhu, Y., Pu, Y.: Parallel Image Processing Based on CUDA, IEEE, Los Alamitos, 2008.

[8] Placido Salvatore Battiato, “High Performance Median Filtering Algorithm Based on NVIDIA GPU Computing,” 2016.

[9] S. Saxena, N. Sharma, S. Sharma, “GPU constructed image segmentation using first order edge detection operators in CUDA environment,” in Journal of Chemical and Pharmaceutical Research, vol. 8(2), 2016, pp. 379 – 387.

[10] Enhua Wu, University of Macau; Youquan Liu, Chinese Academy of Sciences, China; “Emerging Technology about GPGPU”, Circuit and Systems, 2008. APCCAS 2008. IEEE.

[11] Karthik Balasubramanyam, Prabhu, P., Jablin, J., Johnson, N., Beard, S., and august, D. “Automatic CPU-GPU communication management and optimization”, in Proc. of ACM Conference on Programming Language Design and Implementation, 2014.

[12] NVIDIA, "What is GPU Computing?," 2012. http://www.nvidia.com/object/GPU_Computing.html.