International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
International Conference on Industrial Automation and Computing (ICIAC-12-13th April 2014)

RESEARCH ARTICLE OPEN ACCESS

Feasibility Analysis of Bilateral Filtering by General Purpose Graphical Processing Unit Computing

For denoising digital images photographed under low light conditions

Shruti S.Agrawal[1], C.K.Kurve[2]
Department of Electronics Engineering
Kavikulguru Institute of Technology and Science, Ramtek, Dist: Nagpur
[1]shruti.agrawal06@gmail.com

Abstract—
Digital Image Processing is an evergreen area of research in the signal processing domain. Denoising of digital images is one of the most fundamental operations that is performed in the pre-processing stage of almost all image processing operations. This important feature makes denoising as one of the lucrative research areas within the broad area of Digital Image Processing. With the advancement of upcoming hardware technologies, the demand of scientific computing can be better addressed by the image processing researchers.
The clock frequency of a typical CPU has hit a ceiling at around 3.5 GHz, and the trend of the CPU market is now shifted towards providing more numbers of processing cores rather than higher clock frequencies. Under such scenario, it becomes necessary for a programmer to develop parallelized software algorithms, to fully exploit the multicore features of the available hardware The latest trend in advanced computation platforms is the General Purpose Graphical Processing Unit (GPGPU) computing, where the conventional use of GPU's for gaming purpose alone is extended greatly, and exploited to solve general engineering problems of high dimensionality. As long as any problem has the ability to be parallelized, it can be optimized for implementation on the GPU.
Therefore, the use of GPU for implementation of the bilateral filtering [1] algorithm is proposed, since the latter is an inherently parallelizable algorithm. As a practical application, the algorithm is proposed to be used for analyzing its denoising of digital images photographed in low light conditions. Preliminary results for Bilateral Filtering implementation using Matlab strongly favor the further development of this approach for the said application [3].
Index Terms—General Purpose Graphical Processing Unit Computing, Bilateral Filtering, Digital Image Processing, Multicore Processing

I. INTRODUCTION

Digital cameras have become a part and parcel of every person's life. Cameras with a reasonably large density of pixels are available not only as dedicated devices, but also integrated in mobile phones. However, the common problem with majority of photography situations is insufficient lighting. Distortion in the images captured in low light primarily consists of grainy pixels. Before any other image enhancement technique is implemented, it is first necessary to pre-processes this image in order to remove this granular distortion.
The bilateral filtering [2] is an edge preserving smoothing technique, which can be implemented to solve the above mentioned problem. However, as discussed in Section IV, its implementation is simple yet slow. To make this useful for processing high resolution images, an enhancement in the computation speed would be very favorable. A lot of research is already done on the modification of the conventional bilateral filtering algorithm [4]. We propose a hardware based modification approach to the conventional bilateral filtering algorithm, by using the General Purpose Graphical Processing Unit Computation. This can greatly enhance the practical usability of the Bilateral Filtering algorithm.

II. THE GPGPU COMPUTATION PLATFORM.

General purpose graphics processing unit, GPGPU, is the utilization of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU).

The Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model created by NVIDIA and implemented by the

graphics processing units (GPUs) that they produce. CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest Nvidia GPUs become accessible for computation like CPUs. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving general-purpose (i.e., not exclusively graphics) problems on GPUs is known as GPGPU.

On installation of the CUDA toolkit (ver. 4.2) for Windows, it automatically integrates with a pre installed Microsoft Visual C++ (VC++) in Visual Studio Express 2010. Thus, the user friendly environment of VC++ becomes the platform for writing code that can run on the GPU. The CUDA toolkit contains nvcc compiler, which compiles the portions of the code to be executed on the GPU, whereas the rest of the code is compiled by the VC++ Compiler.

Owing to this simplification of programming procedure, the advantages of the GPU Computing can be exploited easily by a programmer not very well experienced with the GPU architecture. Thus the entire focus can be laid on application algorithm development, rather than getting familiarized with the syntax and programming methodologies of the CUDA programming language.

CUDA is the hardware and software architecture introduced by NVIDIA in November 2006 to provide developers with access to the parallel computational elements of NVIDIA GPUs. The CUDA architecture enables NVIDIA GPUs to execute programs written in various high-level languages such as C, Fortran, OpenCL and DirectCompute. The newest architecture of GPUs by NVIDIA (codenamed 'Fermi') also fully supports programming through the C++ language.

Because the GPU and CPU both serve different purposes in a computer, their microprocessor architecture are very deferent. While CPUs currently have up to eight processor cores, a GPU has hundreds of cores. For example, the NVIDIA Tesla 20-series has 448 CUDA cores.

Compared to the CPU, the GPU devotes more transistors to data processing rather than data caching and flow control. This allows GPU's to specialize in math-intensive, highly parallel operations compared to the CPU which serves as a multi-purpose microprocessor. Therefore, calculations of the FDTD algorithm are potentially much faster when executed on the GPU instead of the CPU. This is becoming increasingly true as graphics card vendors such as NVIDIA and AMD are now developing more graphics card for high performance computing (HPC) such as the NVIDIA Tesla.

CUDA has a single-instruction multiple-thread (SIMT) execution model where multiple independent threads execute concurrently using a single instruction. CUDA GPUs have a hierarchy of grids, threads and blocks. Each thread has its own private memory. Because of advancements in technology, the processing power and parallelism of GPUs are continuously increasing. CUDA's scalable programming model makes it easy to provide this abstraction to software developers, allowing the program the automatically scale according to the capabilities of the GPU without any change in code. This is illustrated in Figure 1.

Each thread has its own private memory. Shared memory is available per-block and global memory is accessible by all threads. This multi-threaded architecture model puts focus on data calculations rather than data caching. Thus, it can sometimes be faster to recalculate rather than cache on a GPU. A CUDA program is called a kernel and the kernel is invoked by a CPU program. The CUDA programming model assumes that CUDA threads execute on a physically separate device (GPU). The device is a co-processor to the host (CPU) which runs the program. CUDA also assumes that the host and device both have separate memory spaces: host memory and device memory, respectively. Because host and device both have their own separate memory spaces, there is potentially a lot of memory allocation, deallocation and data transfer between host and device. Thus, memory management is a key issue in GPGPU computing inefficient use of memory can significantly increase the computation time and mask the speed-ups obtained by the data calculations.

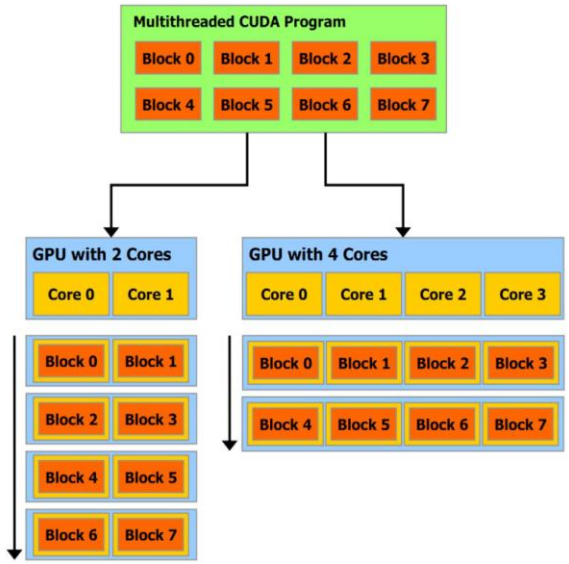

Figure 1 A GPU with more cores will automatically execute the program in less time than a GPU with fewer cores.

III. BILATERAL FILTERING

A bilateral filter is non-linear, edge-preserving and noise-reducing smoothing filter. The intensity value at each pixel in an image is replaced by a weighted average of intensity values from nearby pixels. This weight can be based on a Gaussian distribution. Crucially, the weights depend not only on Euclidean distance of pixels, but also on the radiometric differences. For example, the range difference such as color intensity, depth distance, etc. This preserves sharp edges by systematically looping through each pixel and adjusting weights to the adjacent pixels accordingly.

IV. MATLAB IMPLEMENTATION OF THE BILATERAL FILTERING ALGORITHM

The results of Bilateral Filtering implementation using Matlab are presented for a color image having size of 640x930 pixels. Figure 2 shows a down sampled version of a noisy image photographed with a 55-250 mm telephoto lens mounted on a 12.2 Megapixel Cannon DSLR camera. The shutter speed being mandatorily set to very fast (to capture the sport event), and higher ISO settings to obtain optimum exposure resulted in a grainy image.

On implementing the bilateral filter, the noise is reduced to a great extent, as seen clearly in Figure 3.

The edge preserving nature of the bilateral filter can be clearly seen in Figure 4, which is magnified view of the Figure 2, and comparing it with Figure 5, which is a magnified view of the Figure 3.

The total time elapsed was measured to be 36.24 second using the matlab 'tic' 'toc' commands. This is a reasonably large duration of time, and hence the faster implementation methodology is needed for bilateral filtering.

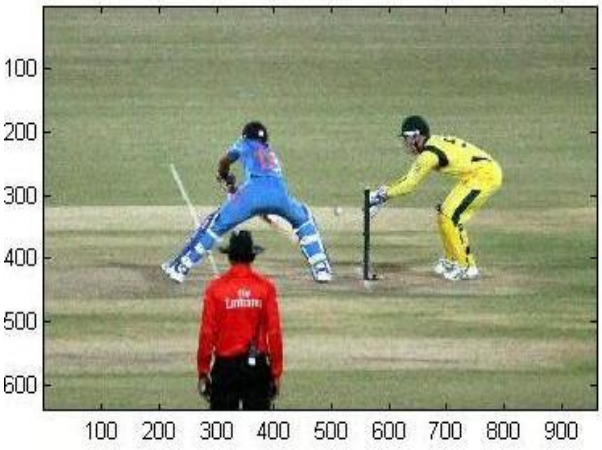

Figure 2: Original digital image with granular noise.

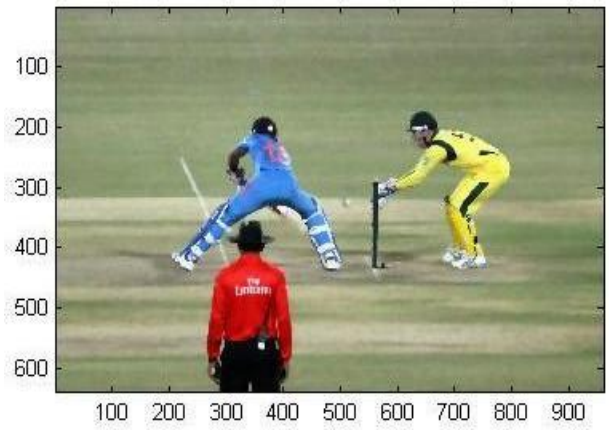

Figure 3: Denoised Image with the Bilateral Filtering Algorithm.

Figure 4: Magnified view of the Figure 2.

Figure 5 : Magnified view of the Figure 3.

V. CONCLUSION

Although the conventional bilateral filtering algorithm dates back to more than a decade old, however with the advent of faster processing techniques like GPGPU Computing, once again bilateral filtering becomes an interesting and useful topic for research and development owing to its parallelizable nature.

ACKNOWLEDGMENT

The author acknowledges the timely guidance and support of Prof. C.K.Kurve.

REFERENCES

[1] Singhal, N. ; Man Hee Lee ; Sungdae Cho ; , "Design and Performance Evaluation of Image Processing Algorithms on GPUs", IEEE Transactions on Parallel and Distributed Systems, (Volume:22 , Issue: 1), January-2011

[2] Agarwal, D. et. al, "Acceleration of Bilateral Filtering Algorithm for Manycore and Multicore Architectures" at Parallel Processing (ICPP), 41st International Conference, Sept 2012

[3] Xiangdong Zhang et. al, "Enhancement and noise reduction of very low light level images" IEEE 21st International Conference on Pattern Recognition, Tsukuba 2012.

[4] "Bilateral Normal Filtering for Mesh Denoising" from Youyi Zhenget.al. at Visualization and Computer Graphics, IEEE Transactions on (Volume:17,Issue:10)