# Fast omnidirectional image un-WRAPPING on gpu

Ilker GÜRCAN, Eldjon KEPUCKA and Alptekin TEMIZEL

Graduate School of Informatics, Middle East Technical University  
Inonu Bulvari, 06531, Ankara, Turkey  
Emails: [Ilker GURCAN](mailto:gurcan.ilker@gmail.com) [Eldjon KEPUCKA](mailto:eldjonkepucka@gmail.com) [Alptekin TEMIZEL](mailto:atemizel@metu.edu.tr)  
web: [Informatics Institute, METU](http://ii.metu.edu.tr)

## Abstract

*The transformation of Omni-directional (OD) images into perspective-view images is an intensive task which takes significant time especially when the image resolution is high. Recently, Graphics Processing Units (GPU) have been started to be used for process intensive tasks, including various image processing algorithms, and significant speed-ups against CPU counterparts have been reported. In this paper, we propose parallel implementation of un-warping OD images on GPU, implemented using Compute Unified Device Architecture (CUDA). The performance of the proposed parallel algorithm has been analysed on two different GPU devices against the performance of its sequential version executed on the CPU. Our test results indicate a speed-up of up to 32 times by the execution of the parallel algorithm on the GPU. The performance analysis also indicates that the memory copy operations take significant amount of time on the GPU implementation. Hence we further analysed these copy operations and suggested optimizations for them.*

### INTRODUCTION

Omni-directional (OD) cameras offer users a wide field of view of 360 degrees and they have found use in a variety of domains such as robotics, surveillance, geographic based systems [1-5]. The main obstacle arising with OD cameras is related to their circular shape of acquired images which makes them very difficult for a human being to interpret and operate on. Hence, the process of transforming these images into perspective images (image unwarping) is very critical in the OD image processing domain.

A literature review on the OD image processing reveals that the OD image unwarping algorithms are already well developed and various implementations exist [1-4]. Due to the significant computational requirement of these algorithms, they are mostly applied to relatively lower resolution OD images.

Parallel algorithms have successfully been applied in a large number of image processing tasks such as image edge detection [6], sampling, colour transformation, convolution operations, segmentation [7] and background segmentation [8], multi-view stereo matching, linear feature extraction, and JPEG2000 encoding [9]. The notable results achieved on these papers indicate that the GPUs are suitable for a variety of image processing operations. In a recent paper, the authors present an important initial indicator of the potential of GPU device for OD image unwarping [5]. In this paper over 9x speed-up is reported against the CPU implementation. However, as the paper is focussed on not only image unwrapping but on proposing a tracking framework, the details of the unwrapping algorithm and parallelization issues are not discussed in detail. In a parallel GPU implementation, method of thread utilisation, effects of selection of different memory types, interpolation methodologies are important and have direct impact on the performance. Also, as parallel implementations result in different speed-ups for different data sizes (as this might generate different levels of occupation in parallel systems), it is important to analyse the effect of different image sizes on the algorithm performance.

In this paper, we intend to demonstrate the significant speed up by parallelization of current OD image unwarping algorithms on the GPU. We propose technical solutions regarding CUDA memory type selection which as will be shown later on, plays a critical role on performance achievement. We analyse and discuss our results for different memory types, interpolation techniques and image sizes.

### SEQUENTIAL ALGORITHM PARALLELIZATION

A large number of image processing algorithms are based on independent pixel processing which lend themselves well to parallel processing as the parallelization can be done in pixel level. In the next sections we intend to present a very brief picture of GPU architecture with CUDA programming model, OD image unwarping algorithm and demonstrate that the algorithms is suitable to be applied on a GPU device.

**2.1 GPU architecture**

GPU

SM 0

Memory

Cores

SM 1

Cores

Memory

SM n

Cores

Memory

Figure 1: GPU architecture.

GPU represents an innovative architecture designed for supporting computationally-intensive tasks which features fit to CUDA parallel programming paradigm. Its simple structure is one of the strongest advantages of this architecture and provides scalability. In Figure 1 a general GPU architecture is depicted. The core element of a GPU device is called Streaming Multiprocessors (SM). Each SM is compound by two main elements: a set of computational units (cores) and the built-in memory. The GPU device has its own DRAM memory to which we refer as the device memory. Recent architectures include as much as 512 cores organised into 16 SMs and 6GB of device memory.

**Grid**

Block 0

**Host**

Shared Memory

Registers

Registers

Registers

Registers



Thread 0

Thread 3

Thread 2

Thread 1

Global, Constant, Texture Memory

Kernel

Figure 2: CUDA programming model

The high number of cores enriches this architecture with capabilities of parallel execution of a large number of threads. While this highly parallel architecture has significant benefits in terms of computation speed, it requires parallel algorithms.

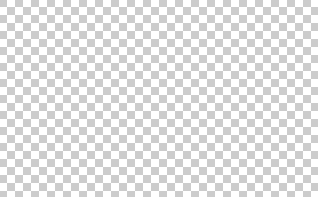
CUDA intends to utilize the GPU computational capabilities by launching tens of thousands of threads simultaneously which will typically execute the same instructions on different set of data. The key point stands on the ability and possibility of thread-data mapping. Due to this reason, CUDA enables threads organization accordingly to the programmer requirements. Threads maybe organized into one, two, or three dimensional arrays called *blocks*. The entire set of blocks is organized into one or two dimensional array called *grid*.

We categorize the CUDA programming strategy into *divide and conquer* consisting on dividing the dataset into *n*-smaller subsets and apply the same algorithms over these subsets, theoretically speeding up the process by *n*-times.

**2.2 OD Image Unwarping Parallel Algorithm**

OD image unwarping sequential algorithm proposed in [1] clearly indicates that each image pixel can be considered as an independent unit, moreover inferring the suitability to CUDA parallelization approach by mapping a pixel-thread tuple. This approach is frequently implemented in image processing parallel algorithms. However, we identified another critical issue arising with our parallelization task.

If we overview the OD image which can be found in figure 3(a) we understand that from the entire image there is a section which stands for the real image captured by OD camera.











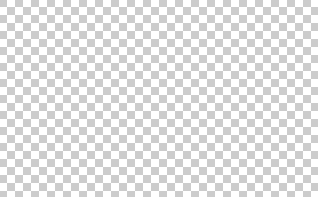


*r*





(a)









(b)

Figure 3: Pixel relation between (a) OD image and (b) Panoramic image.

In 3(a) this section is highlight and it is represented by the area located between the black and blue circle. The *black area* inside the blue circle is a characteristic of OD images due to the limited field of view of OD cameras regarding perpendicular field of view. Therefore, our task consists of unwarping just the grey area and avoids any waste of resources by operating on the rest of the image which indeed does not contain any important information.

We identify two main factors having direct impact on GPU implementation:

* Providing equal task distribution per *thread/warp/block*
* Having coalesced memory access

When implemented on GPU, the OD to panoramic image mapping equations proposed in [1] lead to either an unequal task distribution among *threads/warps/blocks* or un-coalesced memory access. First scenario comes in front in case of a typical mapping of line of pixels to *thread/warp/block* causing the dependency of the completion of the algorithm by the *thread/warp/block* to which was distributed the largest task (the central line of pixels). The second scenario is played if takes place an equal task distribution among *threads/warps/blocks* (mapping of circle sections to *threads/warps/blocks*) causing unnecessary latency due to un-coalesced memory access.

To avoid in a certain scale the scenarios depicted above we propose the reverse mapping equations between OD and panoramic images’ pixels. In other words we intend to map panoramic image pixels to OD image pixels. From figure 3, the relation between points and is given by the following equations:



(1)

Where: and are the coordinates of the pixel on the OD image, and are the coordinates of the pixel in the panoramic image, R and r are respectively radiuses of internal and external circle of OD picture meaningful area, and are the panoramic image sizing parameters, and is the angle between the line (P , O) which passes through the centre of the OD image and pixel P, and the lineof the Cartesian coordinating system.

The equation (1) we propose is relevant for identifying the coordinates of the OD image pixel based on its respective panoramic image pixel coordinates.

**2.3 Interpolation Techniques**

The process of moving from the edge of the real OD image to the centre of it, it is accompanied by an increased number of pixels in the panoramic image without their corresponding pixels in the OD image. Somehow we have to estimate the values of these pixels. We accomplished the estimation process by using Gaussian smoothing filter from OpenCV.

### GPU IMPLEMENTATION

Our implementation strategy is based on two objectives which were described in the previous section, equal job distribution among threads/warps/blocks and fast memory access. The first objective is achieved by implementing OD image unwarping reverse algorithm according to the equation (1).

Each warp has been assigned the processing of one horizontal line of pixels of the panoramic image (destination image). Within the warp, we distribute pixels to threads according to the same approach. In other words, one unwrapping iteration concludes with the processing of 32 pixels of the destination image. The warp runs into a loop until the number of pixels equal to the width of the destination image is being processed. It is understandable that in the width of the destination image is not a dividable by 32 then *32 – modulo (width, 32)* threads will idle.

The second objective achievement is determined by memory types. Referring to figure 2, GPU architecture offers a certain number of memory types including shared, global, constant, and texture memories. From this group shared memory offers the fastest memory, yet it is very limited. Global memory is an off-chip memory which is significantly slower, however much larger than shared memory. Constant and texture memory are read-only cached memory, therefore generally faster than global memory. However, global memory is considered to be faster in case of coalesced memory access.

The focus of this work is to successfully unwarp high-quality OD images. Our main OD image dimensions to be unwrapped are 5184 and 3456 pixels which refer to approximately 68 MB of memory.

Due to the above restrictions we selected texture memory to bind the OD image.

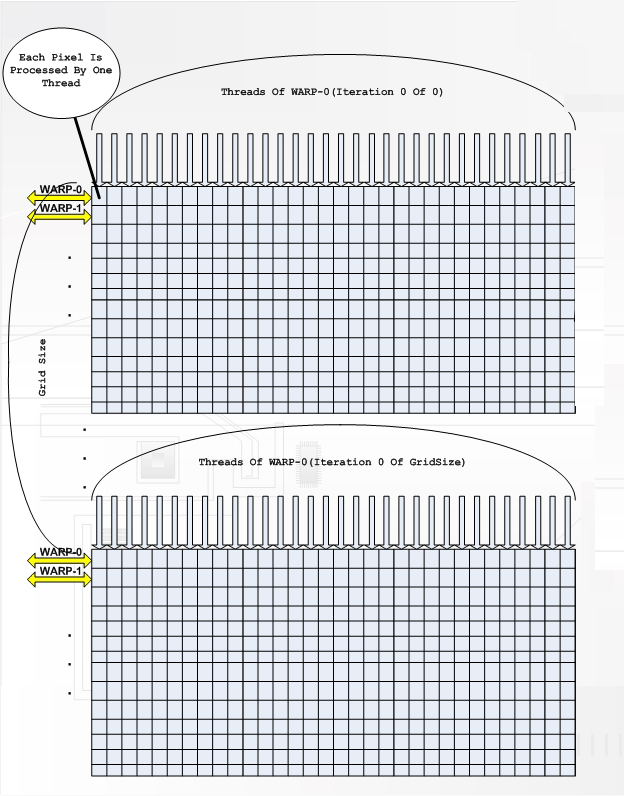


Figure 4: CUDA threads organization

The execution of the OD image unwarping parallel algorithm gives the below result. The parameters and were respectively set to equal one and two. The dimensions of the resulting panoramic image are 10857x1728 and it has three channels (RGB).

### test results

**4.1 Methodology**

OD image unwarping parallel algorithm was implemented in two versions sequential and parallel, to run respectively on a CPU and GPU device. For each single test case the algorithm was executed ten times and the final execution time of each test case stands for the arithmetic average of the ten test executions. For each test case of the parallel algorithm execution we measure separately the time of: host to device memory copy operation, kernel execution and device to host memory copy operation. Total execution time is compared with the sequential algorithm execution time, yet separate timers are indicators of the relevance of memory coping operation in GPU overall performance. The parameters and were respectively set to equal one and two during the entire testing procedure.



(a)



(b)

Figure 5: (a) OD image and (b)its corresponding panoramic image obtained by applying OD image unwarping parallel algorithm

Throughout testing procedures parallel algorithm was executed in two GPU devices: *NVIDIA Quadro FX 5800* and *NVIDIA Quadro 2000* and sequential algorithm were executed on an *Intel Core i7 CPU 2.93 GHz* device.

**4.2 Texture memory vs. global memory**

As expected our test results indicate that texture memory is significantly superior to the global memory in this type of un-coalesced memory access. To compare using texture and global memory, we have run execution time tests for two types of memory usage: Global (G) and Texture (T) and for two types of interpolation techniques: Bilinear and Closest Neighbour. In figure 6, comparisons for these different approaches are given for Quadro 2000 and Quadro FX5800. Overall the results highlight that texture memory outperforms global memory by 38 – 70%. The gain is more significant when the kernel computational complexity is lower (in this case moving from bilinear interpolation to closest neighbour interpolation). The results suggest that it is preferable to use texture memory.

**4.2 Effect of memory copy operations on performance**

One of the important facts highlighted by our test results is the dominance of memory copy operations on the overall execution time. High resolution images exceed tens of MBs, and as the GPU (device) and CPU (host) memory spaces are separate, the image data needs to be copied back and forth. In our tests, the full size input image copied from host to device was 68MB while the output image copied from device to host was 71MB.

Figure 6: Global and texture memory performance.

The ratio of these copy operations and calculation time to the overall execution time is demonstrated in Figure 7 for various implementations. As seen from this figure, memory copy operations are responsible for approximately 78- 95% of the time of the overall execution on the Quadro FX 5800 device. Another implication of this observation is that, as the memory operations are the dominating factor, it might be preferable to use relatively more complex algorithms as the overall execution time is not affected significantly.

Figure 7: Memory copy operations and kernel execution time for NVIDIA Quadro FX5800.

**4.3 Kernel Execution Time for Different Image Size**

For this test case the entire set of images to be processed was obtained from the original image by down-sampling. In Figure 8, the results of parallel OD image unwarping algorithm on texture memory accompanied by bilinear interpolation technique for three different image sizes are shown.

**4.4 CPU vs. GPU**

Finally and the most relevant results we obtained regarding the speed-up of the GPU implementation of the OD image unwarping parallel algorithm over the CPU implementation of its sequential version. Previously we identified the best performing parallel algorithm which was obtained by implementing it on texture memory with the closest neighbour interpolation technique on the NVIDIA Quadro FX 5800 device. For comparison we implemented on C++ the same algorithm and executed it on a single thread environment on an Intel i7 860 CPU.

Figure 8: Algorithm performance for different image size.

Test results confirmed the superiority of the GPU architecture over the CPU in terms of computationally complex tasks. In figure 9 we included the time performance of the algorithms for varies image sizes. In order to highlight the relevance of memory copy operations we display the parallel algorithm execution time into three cases: plain kernel execution time, kernel execution time with host to device memory copy time, and kernel execution time with host to device and device to host memory copy time. With the decrease of the image size, test results indicate two different trends on speed-up gain: decreasing trend for the plain kernel execution time with the speed-up varying from 827 and steadily decreasing to 717, and increasing trend for the total execution time including the memory copy operations where the speed up varies from 30 and steadily increasing to 41.

Figure 9: GPU vs. CPU performance chart and the effect of the memory copy operations.

This is one more case indicating the clear relevance of memory copy operations which practically may dictate the speed-up gain.

### future work

### The results obtained from the GPU implementation of OD image unwarping algorithm are a clear indicator of the potential of the GPU architecture to significantly increase the performance of image processing tasks such as OD image unwarping. Our test results indicate up to 32 times speed up against the CPU version. Besides providing a significant speed up, the GPU implementation makes new applications possible as it makes a real time system possible. While processing an image takes 2.45 seconds (approximately 0.4 images/second), the same process takes 75.82 milliseconds (approximately 13.2 images/second) on GPU. When integrated into a surveillance system, it may lead to the unwarping of real-time video streaming obtained by OD cameras.

### References

[1] H. Liu, Z. Huo, and G. Yang, “Omnidirectional Vision for Mobile Robot Human Body Detection and Localization”, in *Proc.* *IEEE Systems Man and Cybernetics*, Istanbul, Turkey, October 10-13. 2010, pp. 2186-2191

[2]J. Gluckman, S. K. Nayar, and K. J. Thoresz, “Real-Time Omnidirectional and Panoramic Stereo,” in *Proc. DARPA Image Understanding Workshop*, November 1998, pp. 299-303

[3] S.W. Jeng and W. H. Tsai, “Precise Image Unwarping of Omnidirectional Cameras with Hyperbolic-Shaped Mirrors,” in *Proc. IPPR Conference of Computer Vision, Graphics, and Image*. 2003, pp. 414-422

[4]V.N. Peri and S. K. Nayar, “Generation of Perspective and Panoramic Video from Omnidirectional Video,” in *Proc. DARPA Image Understanding Workshop*, New Orleans, USA, May 1997, pp. 243-245

[5] S. Ghidoni, A. Pretto, and E. Menegatti, “Cooperative tracking of moving objects and face detection with a dual camera sensor,” in *Proc. 2010 IEEE International Conference on Robotics and Automation,* Alaska, USA, May 3-7. 2010, pp. 2568

[6] N. Zhang, Y. Chen, and J. Wang, “Image Parallel Processing Based on GPU,”in *Proc. IEEE Advanced Computer Control*, Shenyang, China, March 27-29. 2010, vol. 3, pp. 367

[7] N. Singhal, I. K. Park, and S. Cho, “Implementation and Optimization of Image Processing Algorithms on Handheld GPU,” in *Proc. 17th IEEE International Conference on Image Processing*, Hong-Kong, China, September 26-29. 2010, pp. 4481

[8] A. Temizel, T.Halici, B.Logoglu, T.Taskaya Temizel, F.Omruuzun, E.Karaman, "Experiences on Image and Video Processing with CUDA and OpenCL", NVIDIA GPU Computing Gems (Elsevier), vol. 1, Jan. 2011.

[9] I. K. Park, N. Singhal, M. H. Lee, and S. Cho, “Efficient Design and Implementation of Visual Computing Algorithms on the GPU,” in *Proc. 16th IEEE International Conference on Image Processing*, Cairo, Egypt, November 7-10. 2009, pp. 2321