**Time Reduction in Video Compression Using the Sum of Absolute Differences Algorithm in CUDA**

*Shruti Vangari, Merin Jose, Kashyap Uppuluri  
Department of Computer Science, University of Akron*[*sv47@zips.uakron.edu*](mailto:sv47@zips.uakron.edu)*,* [*mj65@zips.uakron.edu*](mailto:mj65@zips.uakron.edu)*,* [*ku10@zips.uakron.edu*](mailto:ku10@zips.uakron.edu)

**Abstract**

*This paper gives a new insight into the computation of the Sum of Absolute Differences Algorithm for Video Compression. We implemented the SAD algorithm using various techniques in CUDA to improve the time taken for computing the SAD value. These techniques were then compared to the MATLAB simulated code and the conventional CPU code. We compared all the computational times and concluded that CUDA techniques gave a new exemplary dimension to the field of computing. In the near future, millions of processors could be used for computing at once transforming computation speed from seconds to nanoseconds.*

**1. Introduction**

Digital video compression entails the utilization of many coding techniques with the ultimate goal to reduce the size of the digital representation of a video sequence. In video coding, the fact that subsequent video frames do not differ much can be similarly exploited in order to increase compression and computation efficiency.

CUDA - Compute Unified Device Architecture exploits the fact that multiple processors can produce efficient results than a single processor. A conventional CPU code performs sequential execution of all the instructions consuming more time. The world’s first GPU (Graphics Processing Unit), GeForce 256 was introduced by NVIDIA. It was invented to overcome the challenges thrown by the conventional CPU architecture by executing instructions in parallel.

This paper helps us understand the challenges thrown by CPU computing and solves the same using GPU architecture. We can conclude that for very large datasets, GPU implementation reduces cumbersome work and boosts the execution time by a huge factor (1000 in this case).

**2. Design**

A video is composed of a set of frames. Two consecutive frames have a high degree of correlation and this is utilized for video compression. While transmitting the video, only the first frame and the difference between every consecutive frame are sent, thereby reducing the overall data transmitted. Hence, the transmission bandwidth is reduced and compression is achieved.

**2.1 Mathematical Representation of the SAD Algorithm**

The complete SAD operation can be written as:



With 0 ≤ x, y < frame size   
With A(x, y) being a current frame pel at (x, y)   
With B(x, y) being a reference frame pel at (x, y)

Figure 1: Mathematical Formula for SAD

* 1. **Pictorial Representation of SAD**

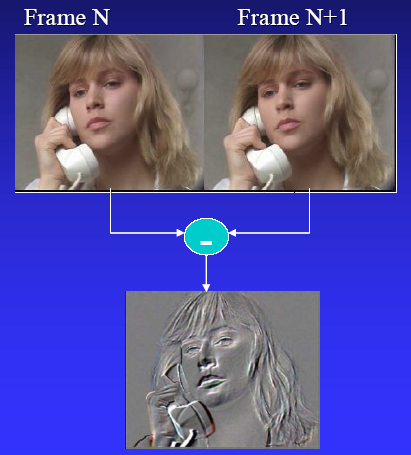


Figure 2: Pictorial Representation of SAD

Consider the above two consecutive frames. Since there is a high degree of correlation between the two frames, we need not transmit both the images. Only the difference is computed and SAD is used as a metric to represent this redundancy. The difference image as shown above represents only the slight change in motion between the two images. The absolute value is taken and the corresponding absolute differences and summed. This gives the SAD value.

First, the columns of the difference image are added and this represents the corresponding SAD value. The lower the SAD value, it implies the two images are similar in that pixel area. A SAD value of 0 represents that the two images are identical. The other SAD values are assessed based on a threshold and then if the SAD values are higher than the threshold, it implies that the two images are entirely different and hence, that data has to be transmitted.

SAD is a metric, usually used in motion estimation and related algorithms. Here, we use it to indicate the redundancy of the pixels so that similar ones can be eliminated and only those which differ by a considerable amount need to be transmitted.

**3. Implementation**

**3.1 Steps to perform Video Compression**

A 1:24 seconds long video was taken and divided into 27 frames using Pinnacle Studio as shown in Figure 3.

Figure 3: Division of the Video into Frames

**3.2 Implementation in MATLAB**

The video is now taken and implemented in MATLAB as shown in Figure 4.

Read Images 1, 2   
(m \* n \* o)

Convert the 3D Images to 2D

(m \* n \* o) 🡪 (m \* n)

Reshape the matrices to columns

(m \* n) 🡪 (mn \* 1)

Take the absolute difference

Convert gray scale to binary values

Write these binary values to a text file

Output = Compute sum of elements of difference file

Figure 4: Flowchart for the SAD computation in MATLAB

**3.2 Implementation for the CPU Code**

Read Image 1 in MATLAB

Read Image 2 in MATLAB

Convert Image 2 to Pixel Matrix and write to a text file 2

Convert Image 1 to Pixel Matrix and write to a text file 1

Read Matrix 2 from text file 2 in CUDA

Read Matrix 1 from text file 1 in CUDA

Difference Matrix

Sum of elements in the Difference Matrix

Figure 5: CPU code flowchart for SAD computation

Here, the files are read and the Difference matrix and Sum is computed in the main function itself.

**3.3 Implementation for the GPU Code**

Read Image 1 in MATLAB

Read Image 2 in MATLAB

Convert Image 2 to Pixel Matrix and write to a text file 2

Convert Image 1 to Pixel Matrix and write to a text file 1

Convert Image 2 to Pixel Matrix and write to a text file 2

Convert Image 1 to Pixel Matrix and write to a text file 1

Read Matrix 2 from text file 2 in CUDA

Read Matrix 1 from text file 1 in CUDA

Apply Technique

Difference Matrix

Sum of elements in the Difference Matrix

Figure 6: GPU code flowchart for SAD computation

Here, the kernel implements the computation of the Difference Matrix and sums up all the elements to compute the SAD value. Various techniques can be applied to the CUDA code to enhance the performance and execution time which increases the efficiency to a 1000 fold.

**4. Results**

To interpret the performance of the GPU, we had compared the results produced by the MATLAB Code, the CPU Code, the GPU Code with the Thread concept and the GPU code with tiling. We had implemented these 4 various forms of the SAD algorithm for 13 matrices generated by the 27 consecutive frames of the 1:24 seconds long video.

To generate these matrices, we had padded 2 consecutive frames of 2 images to ensure that the data was taken uniformly.

**4.1 Using SAD in MATLAB**

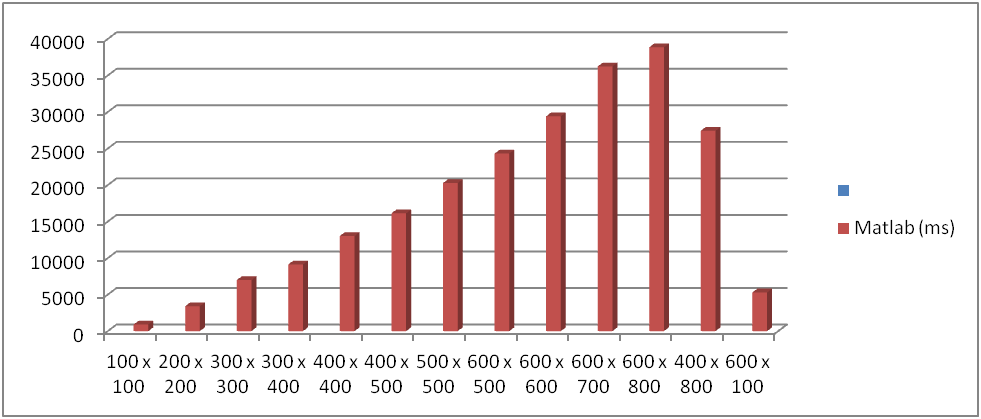
****

Figure 7: Computational time (ms) vs. Matrix Size for the MATLAB code

**4.2 Using CPU Code**

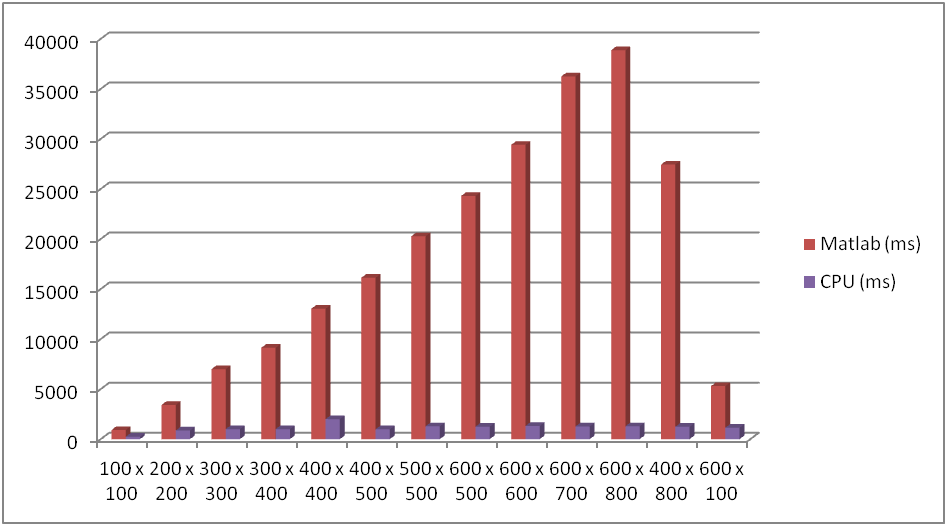


Figure 8: Comparison of the Computational time (ms) vs. Matrix Size for MATALB and CPU codes

**4.3 Using GPU Code**

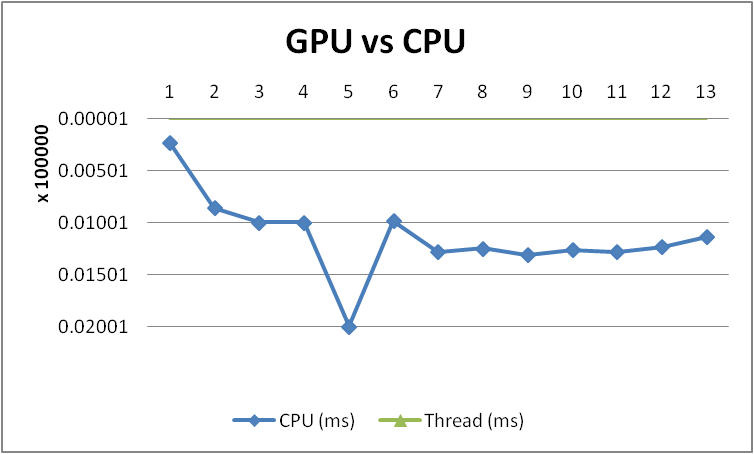
****

Figure 9: Comparison of CPU vs. GPU performance

**4.4 Comparison of all the techniques used**

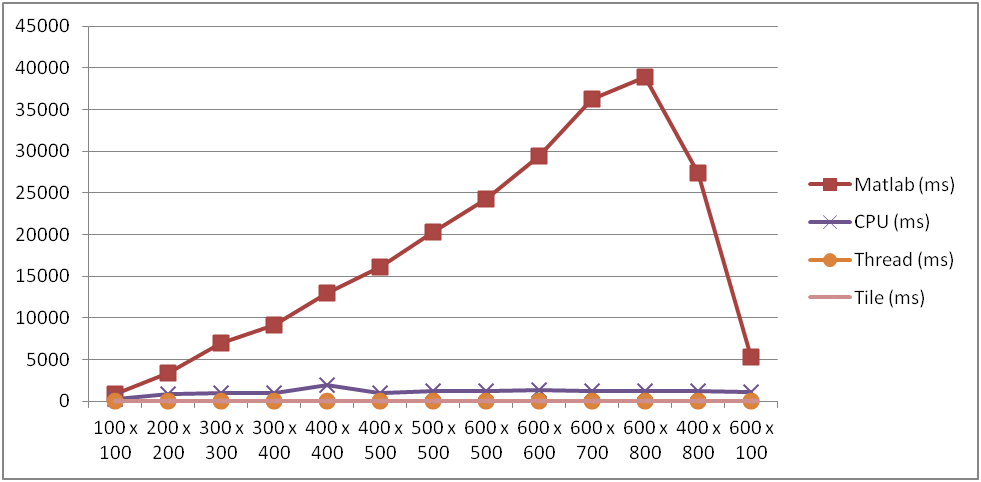
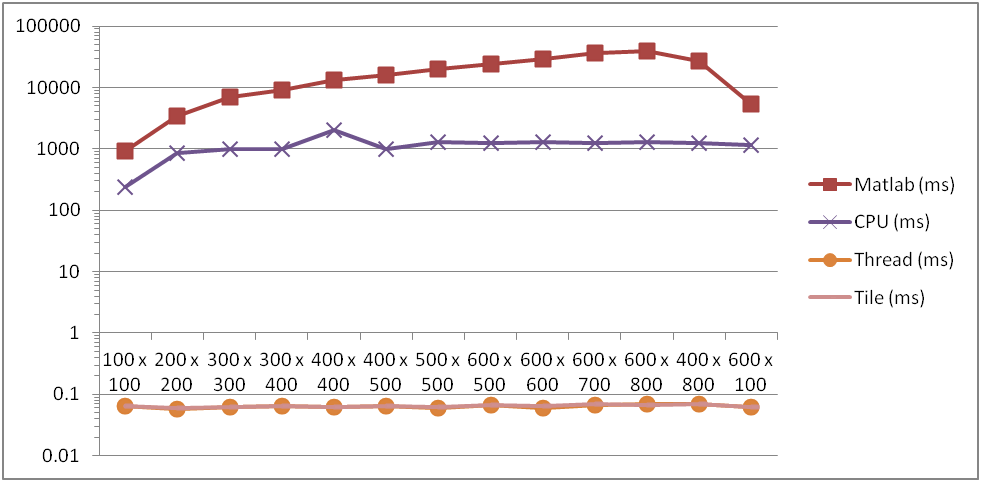
****

Figure 10: Comparison of all the techniques shown as Matrix size vs. Computational time (ms)

For a better understanding, the computational time on the logarithmic scale gives a clear understanding of the performance as shown in Figure 10.

  
Figure 11: Comparison of all the techniques shown as Matrix size vs. Computational time (ms) on a log scale

**5. Conclusion**

The computation of the GPU gives the best performance using tiling. One could also use other techniques to give an improved performance. We tried implementing the tiling concept with various tile sizes but since the efficiency was already exploited to the maximum limit, the decrease in computational time was negligible.

This algorithm cannot implement the shared memory concept since the memory is limited to 64KB where as the matrix sizes are very large due to the fact that they are images. Hence, the concept of shared memory hence cannot be exploited. The other techniques that follow are Data Prefetching – which is again dependent on the Shared Memory concept cannot be applied here.

The further work that can be done is to explore further techniques that can be used in CUDA. Except for the Shared Memory concept, one can try using other techniques to enhance performance further. The concept of memory coalescing can also be applied since the matrices are implemented in the Row – major concept in CUDA. In general the total efficiency acquired was more than a thousand fold.

**6. References**

[1] D. Chaikalis, N. P. Sgouros, and D. Maroulis. 2010. A real-time FPGA architecture for 3D reconstruction from integral images. *J. Vis. Comun. Image Represent.* 21, 1 (January 2010), 9-16. DOI=10.1016/j.jvcir.2009.09.004 <http://dx.doi.org/10.1016/j.jvcir.2009.09.004>

[2] Brian M. Li and Philip H. Leong. 2008. Serial and Parallel FPGA-based Variable Block Size Motion Estimation Processors. *J. Signal Process. Syst.* 51, 1 (April 2008), 77-98. DOI=10.1007/s11265-007-0143-9 <http://dx.doi.org/10.1007/s11265-007-0143-9>

[3] Sum of Absolute Differences – a journal by Motorola

[4] Wikipedia <http://en.wikipedia.org/wiki/Graphics_processing_unit>

[5] David B. Kirk and Wen-mei W. Hwu. 2010. *Programming Massively Parallel Processors: A Hands-On Approach* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.