American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)

ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

Execution Speed up of Image Rotation Matrix Using Parallel Technique

Dr. Fahraldeen Aldulaimi[a]*, Hadeel Alshakargy[b]

[a,b]*Mosul University, Dept. of Computer Engineering, Erbil 44001, Iraq*

[a]*Email: fhali_a@yahoo.com*

[b]*Email: hadeelalshakargy@yahoo.com*

Abstract

In computer graphic science rotating a vertex in an image around a specific point in any direction is a time consuming mission. The rotation of a vertex depends on multiplying it's coordinates by graphic geometric transformation matrices, this multiplication requires a considerable time. In this paper the acceleration of image rotation is achieved by using parallel techniques such as using Multicore Core Central Processing Unit (CPU) or General Purpose Graphic Processing Unit (GPGPU) or even both. The results show a significant increase in computation speed when rotating a large number of vertices by using CPU. A considerable acceleration is achieved when GPU is used to make image rotation. However the speedup is limited by the number of processing units available for parallel processing.

Keywords: vertices; CPU; Central Processing Unit; GPGPU; General Purpose Graphic Processing Unit.

1. Introduction

Vertex rotation of a shape around any point in a direction specified by a rotation angle can be achieved by applying a graphic geometric transformation matrix on its coordinates (x, y). This rotation depends on the multiplication between the matrix of vertices and the graphic transformation matrix [1,2,12]. Typically a large number of vertices contribute on the execution time. Many attempts are made to reduce execution time taken for multiplication.

---

* Corresponding author.

Recently multicore and multithreaded CPUs with shared memory are a cost effective way of obtaining significant increases in CPU performance. An exponential growth in performance was expected from more hardware threads and cores per CPU [17]. In the other way there are some attempts to speeding the operation of multiplication by Graphical Processing Unit (GPU) [6,10]. In image rotation the methods used for speeding up is depending on the parallel matrix multiplication, rotate all vertices using the rotation matrix simultaneously by assigning each group of vertices to each thread and multiplying them. In this paper accelerating the image rotation is implemented by using core i3 processors and the GeForce GT 635M with 96 core. Most of the nowadays laptops and pcs are provided with the above mentioned parallel platforms. The goals of this paper are to investigate the acceleration of rotation using such platforms under popular parallel programming paradigm MATLAB used in technical computing [11]. Keeping in mind if the performance of MATLAB is inadequate, then there is a need to other programming language such as Visual Studio which may give better results for reducing the execution.

2. Geometric transformation matrix

To rotate a vertex around any point in any direction its coordinates matrix must be multiplied by a combination matrix consists of translation matrix and rotation matrix. Translation matrix is used to transform center of rotation to origin. Rotation matrix contains the angle of rotation. After rotation the transformation matrix is reversed using a third matrix [2,12].

2.1. Rotation matrix

Rotation matrix is one of the graphic geometric transformations applied to each individual vertex and repeated to each of the vertices to achieve the required rotation. The rotation is applied to a vertex by repositioning it along a circular path in (x, y) plane in clockwise or anti clockwise direction specified by an angle.

2.2. Translation matrix

Translation matrix is one of the graphic geometric transformations and is also applied to an individual vertex and repeated to each of the vertices. It is applied to a vertex by repositioning it along a straight line path from one coordinates to another, the translation is applied to each vertex adding the (tx to x) and (ty to y) so that vertex coordinates are changed from V(x, y) to V'(x', y'), where tx and ty are moving distances.

3. Matrices representation

The general forms of rotation and translation matrices are represented as in the following articles.

3.1. Rotation matrix about origin

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (1)$$

3.2. Translation matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2)$$

3.3. Concatenation between Rotation and Translation

Producing a general matrix form to rotate a vertex around any center of rotation can be achieved by multiplying transformation matrix by rotation matrix and the by translation matrix again (See equation (3)).

The outcome of this multiplication is a single concatenation matrix which can be used to computes a new vertex, by making a single matrix multiplication rather than three. The form of concatenation matrix is desecrated in equation (4).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\,\theta & -sin\,\theta & tx(1-cos\,\theta)+ty\,sin\,\theta \\ sin\,\theta & cos\,\theta & ty(1-cos\,\theta)-tx\,sin\,\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4)$$

4. Implementation platforms

The CPU and GPU are chosen in this paper as platforms for implementing vertices rotation around any point for sequential and parallel execution, a brief introduction of each platform is overstated.

4.1. Central Processing unit (CPU)

CPU architecture has only one processing unit in the chip (See figure (1)), for performing arithmetic or logic operations. At any time only one operation can be performed [14].

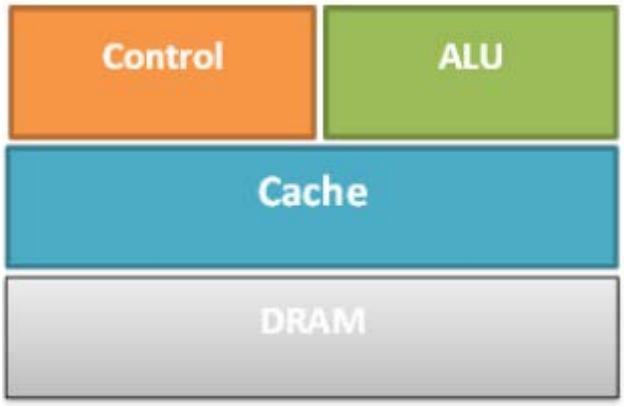

Figure 1: CPU hardware architecture

4.2. CPU with multicore processor

A multicore processor is a system that comprises of two or more independent cores (or CPUs). The cores are generally integrated onto one integrated circuit die (known as a chip multiprocessor), or they are integrated onto multiple dies on a single chip package [17], (See figure (2)).

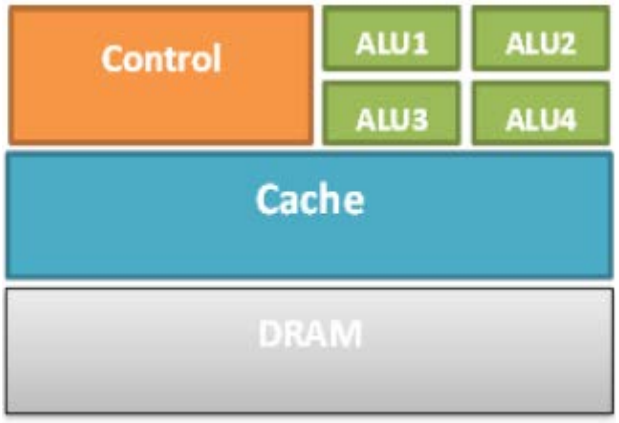

Figure 2: Multicore hardware architecture

4.3. Graphic Processing Unit (GPU)

GPU is viewed as a compute device operating as a coprocessor to the main processor (CPU host). A GPU is implemented as an aggregation of multiple processor so it is called multiprocessors, which is consists of a number of Single Instruction Multiple Data (SIMD) ALUs integrated as a network on a chip (See figure (3)). According to the SIMD every processor within GPU must execute the same instruction at the same time, only data can be varying [11,15,16].

Figure 3: GPU hardware architecture

Refer to figure (3), the orange color indicates the cache memories, the blue color indicates the control units and

the green color indicates the ALUs.

In this paper the image rotation is implemented using laptop of an Intel® Core™ i3-3011 CPU @ 2040 GHZ (4CPUS), ~2.4GHZ, 4MB memory.

And the GPU is GeForce GT 635m version 2, which has 96 cores or shadier processing units (SP), and two streaming multiprocessor units (SM),with 2GB memory. The MATLB and Visual Studio environment have been used to implement software for sequential and parallel of execution. The architecture of GeForce GT 635m in terms of how blocks and threads are arranged as shown in figure (4).

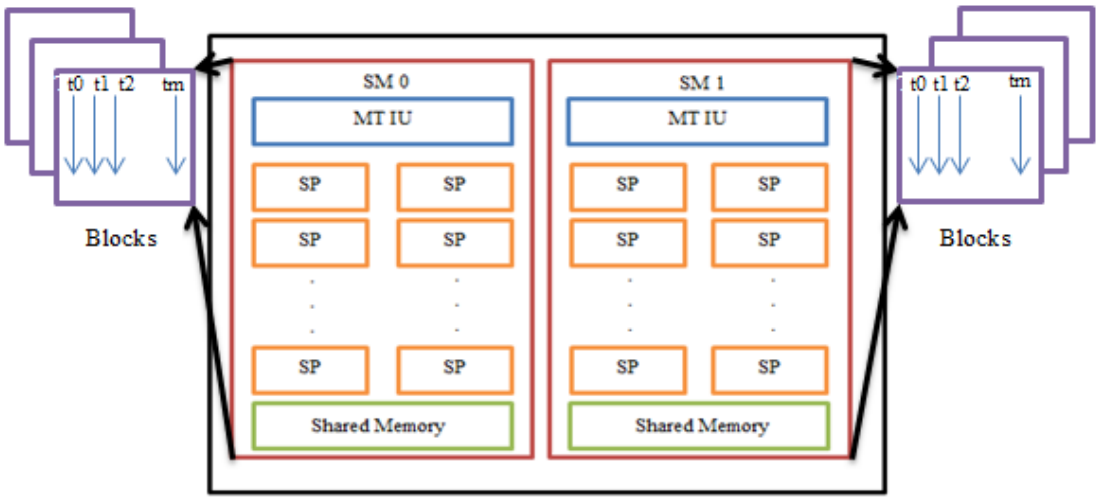

Figure 4: GeForce GT 635m hardware architecture

Refer to figure (4), the two streaming multiprocessor have been represented in SM0 and SM1. The shadier processing units are represented in SPs which represents blocks in software and also represents cores in hardware. The number of shadier processing units is 96 distributed on two streaming multiprocessor units each SM has 48 shadier units as shown in figure (4).

Each SM has a shared memory and multithread instruction unit, each block have a set of threads from t0 to tm in GeForces GT 635m the optimum number of threads has been conclude to be 256 threads per block.

5. Execution and Results

Explain each MATLAB and Visual Studio results on different sets of data.

5.1. MALAB experiment results

These results explain the time of CPU with serial execution and GPU with parallel execution and show the speed factor between them. If the number of vertices exceeds the number of blocks and threads of GPU the time begins to increase exponentially as shown in figure (5) and table (1).

Table 1: Contains the vertices and GPU execution time in second

Vertices	Tile=1	Tile=10	Tile=50	Tile=100	Tile=500	Tile=1000
10	0.0059	0.0059				
50	0.0059	0.0059	0.0059			
100	0.0059	0.0059	0.0059	0.0059		
500	0.006	0.0059	0.0059	0.0059	0.0059	
1000	0.0061	0.0059	0.0059	0.0059	0.0059	0.0059
5000	0.0072	0.006	0.0059	0.0059	0.0059	0.0059
10000	0.0083	0.006	0.006	0.0059	0.0061	0.0059
50000	0.0139	0.0071	0.0072	0.007	0.0071	0.007
100000		0.0075	0.0075	0.007	0.0071	0.0071
500000		0.0111	0.0111	0.0081	0.0093	0.0088
1000000			0.0154	0.0099	0.0117	0.0109
5000000				0.023	0.0324	0.0278
10000000					0.0588	0.0495
50000000						0.3655

Tile means the number of vertices per block, threads per block is (Tile*2) because each vertex consists of (2*1) matrix for x and y coordinates, and blocks per grid is (vertices /Tile). The results are shown that when Tile =1 this means that only one vertex in each block where the execution time is increased when the number of vertices exceeds 100 because the number of blocks in software represents the number of cores in hardware which is equals to 96 core. So when Tile=10 this means that each block contains 10 vertices, from figure (5) the execution time is stabled until 1000 vertex, because 100 blocks each with 10*2 threads are used and this in the range of GPU capacity, and when exceeding this capacity the execution time increases exponentially. And so on for all tiles.

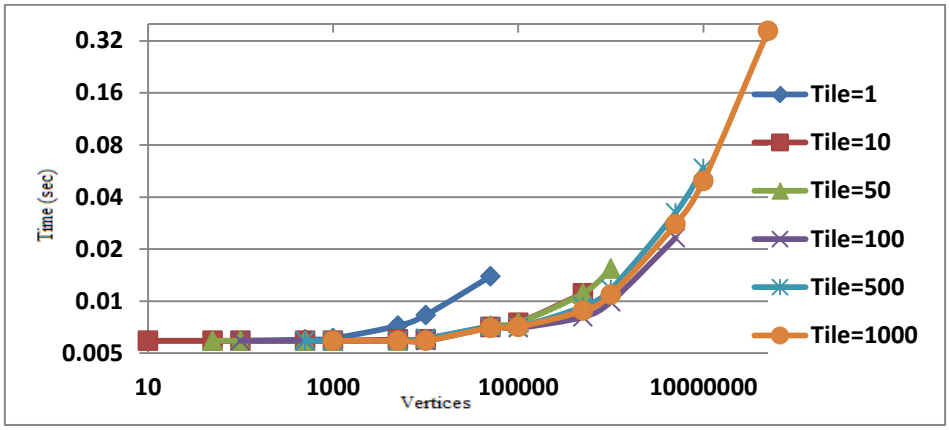

Figure 5: Represents table (1)

Tentatively conclude from these results, when the number of blocks exceeds 100 the number of threads per block exceeds 100 the execution time begins to increase as the number of vertices increase. Then conclude from table (1) that the number of threads per block occurs between 100 to 500 threads. The result shown above does not give the optimum exploitation of GPU, and does not explain the maximum number of threads per block for GeForce GT 635m.

5.2. MATLAB results to determine the actual number of blocks and threads of GeForce GT 356m

As the number of cores in GeForce GT 635m is 96 so multiples of 96 are used in this result to find the number of threads per block. Note that the number of blocks is equals to the number of cores. See table (2) and figure (6).

Table 2: Contains the vertices and the GPU execution time in second

Vertices	Tile=1	Tile=2	Tile=4	Tile=8	Tile=16	Tile=32	Tile=64	Tile=128	Tile=256
12	0.0059	0.0059	0.0059						
24	0.0059	0.0059	0.0059	0.0059					
48	0.0059	0.0059	0.0059	0.0059	0.0059				
96	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059			
192		0.0059	0.0059	0.0059	0.0059	0.0059	0.0059		
384			0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	
768				0.0059	0.0059	0.0059	0.0059	0.0059	0.0059
1536					0.0059	0.0059	0.0059	0.0059	0.0059
3072						0.0059	0.0059	0.0059	0.006
6144							0.0059	0.0059	0.0062
12288								0.006	0.007
24576									0.0075

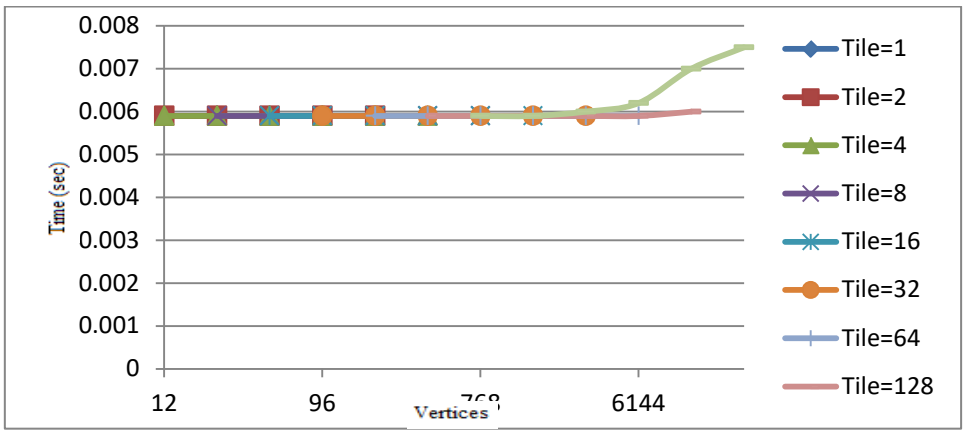

Figure 6: Represents table (2)

From this result it is concluded that the number of blocks is 96 which is equal to the number of GPU cores, and the number of threads per block is 256 and possible to extend to 512 threads.

5.3. visual studio results

This result explains the capacity of GPU (blocks and threads) as shown in figure (7) and table (3).

The same set of vertices in table (1) is used to find the results with Visual Studio.

Table 3: Contains the vertices and the GPU execution time in millisecond

Vertices	Tile=10	Tile=50	Tile=100	Tile=500	Tile=1000
10	0.011936				
50	0.013056	0.012608			
100	0.013152	0.013056	0.0126		
500	0.026976	0.013824	0.013472	0.015264	
1000	0.041632	0.019456	0.0152	0.015456	0.012
5000	0.161472	0.04944	0.044257	0.046592	0.049472
10000	0.314144	0.08928	0.077344	0.085664	0.091232
50000	1.589888	0.405632	0.351392	0.40216	0.43656
100000	3.17184	0.799008	0.692864	0.869606	0.862652
500000	15.812	3.944736	3.419872	4.004646	4.28912
1000000		7.880192	6.82928	8.6916	8.588
5000000			34.10	40.2046	42.88314
10000000				86.9606	85.99
50000000					428.8314

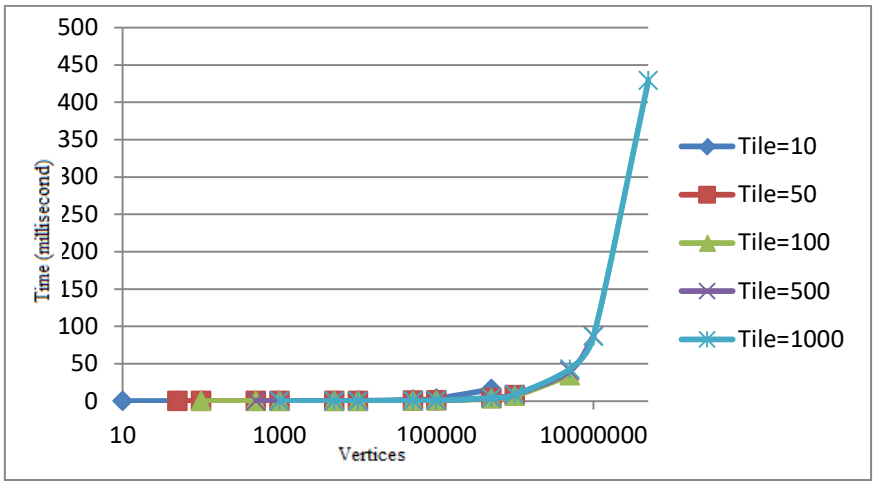

Figure 7: Represents table (3)

5.4. The actual number of blocks and threads in GeForce GT 635m are determined from these results

Table 4: Contains vertices and GPU execution time in millisecond

Vertices	Tile=1	Tile=2	Tile=4	Tile=8	Tile=16	Tile=32	Tile=64	Tile=128	Tile=256
12	0.012032	0.012	0.011872						
24	0.016864	0.0127	0.011267	0.011808					
48	0.02148	0.016576	0.012544	0.012416	0.011776				
96	0.02231	0.021952	0.017152	0.012544	0.012416	0.012384			
192		0.036128	0.022172	0.017216	0.012576	0.012608	0.012544		
384			0.036384	0.022144	0.017248	0.012768	0.012768	0.012864	
768				0.03648	0.022208	0.017728	0.014016	0.013664	0.014112
1536					0.036448	0.02336	0.019232	0.016768	0.016832
3072						0.038112	0.028032	0.02624	0.025982
6144							0.04688	0.044672	0.044384
12288								0.080384	0.08352
24576									0.156224

Refer to figure (8) and table (4), it became obvious that the number of blocks in GeForce GT 635m is 96 blocks per grid and the number of threads is 256 threads per block.

Execution time begins to increase exponentially when exceeding this boundary.

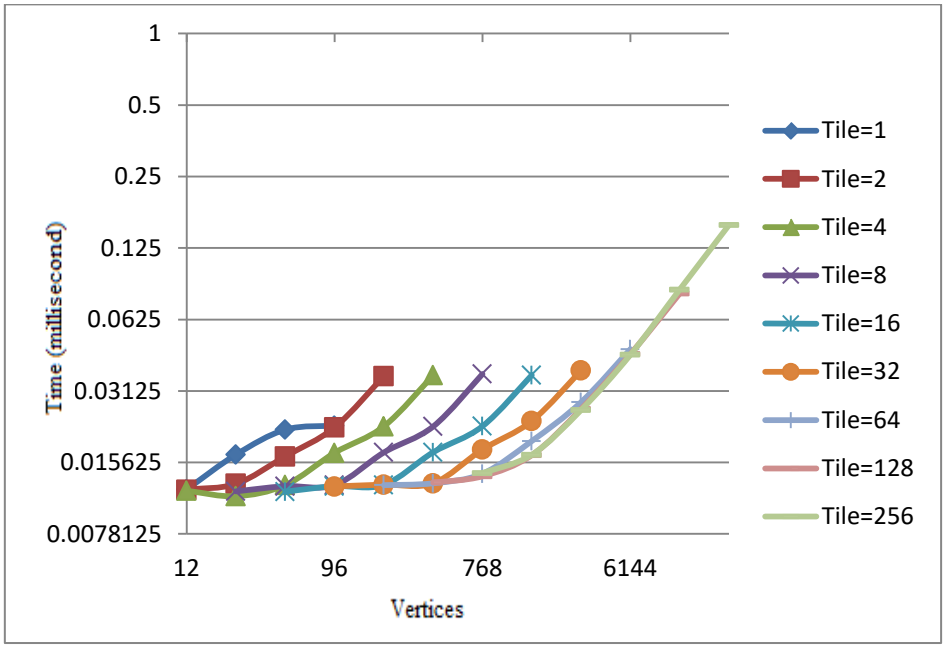

Figure 8: Represents table (4)

5.5. Comparison between CPU and GPU in execution times using MATLAB

CPU execution time in MATLAB is shown in table (5) and figure (9) which is sequential execution.

Table 5: CPU execution time in second

Vertices	Time in sec
10	0.0112
50	0.0121
100	0.0127
500	0.0286
1000	0.0394
5000	0.0591
10000	0.2284
50000	0.6266
100000	1.0518
500000	3.6754
1000000	8.8356
5000000	35.1301
10000000	68.0995
50000000	390.7681

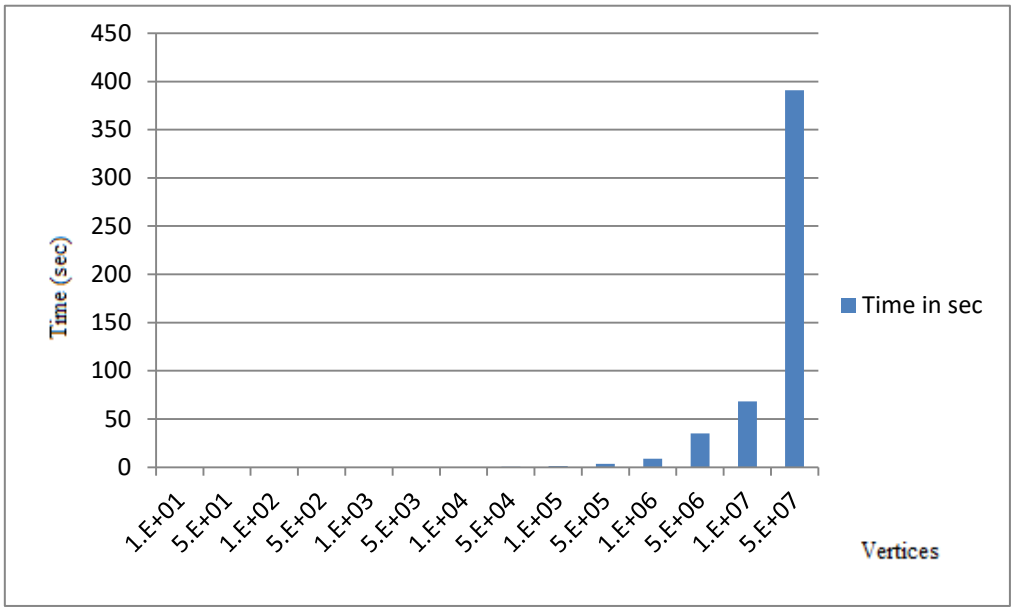

Figure 9: Represents table (5)

CPU execution times for another set of data see table (6) and figure (10).

Table 6: CPU execution time in second

Vertices	Time in second
12	0.0112
24	0.0113
48	0.0121
96	0.0286
192	0.0394
384	0.0591
768	0.0723
1536	0.0852
3072	0.1526
6144	0.2123
12288	0.4261
24576	0.5482

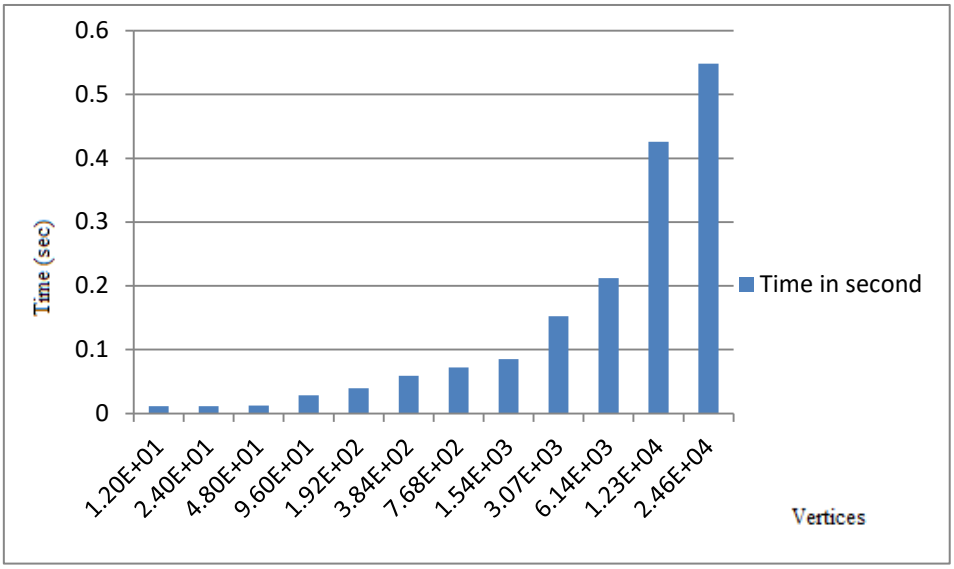

Figure 10: Represents table (6)

GPU execution time in second is taken from table (2).

5.6. Comparison between CPU and GPU in execution times using Visual Studio

CPU sequential execution time using Visual Studio is shown in table (7) and figure (12).

GPU execution time in millisecond has been taken from table (4). See table (8) and figure (13).

Table 7: CPU execution time in millisecond

Vertices	Time millisecond
12	10
24	10
48	16
96	16
192	47
384	156
768	234
1536	308
3072	483
6144	842
12288	1513
24576	2964

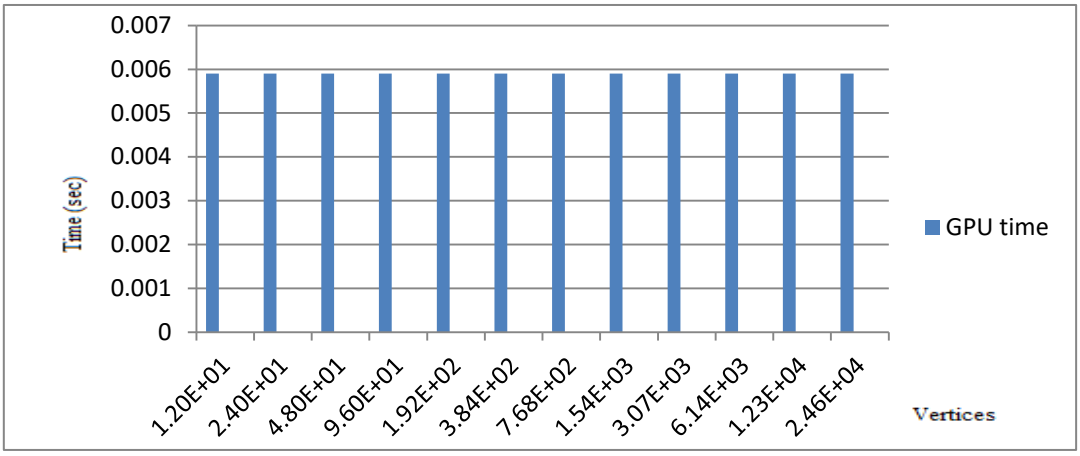

Figure 11: Represents GPU execution time when all bocks and threads have been exploited

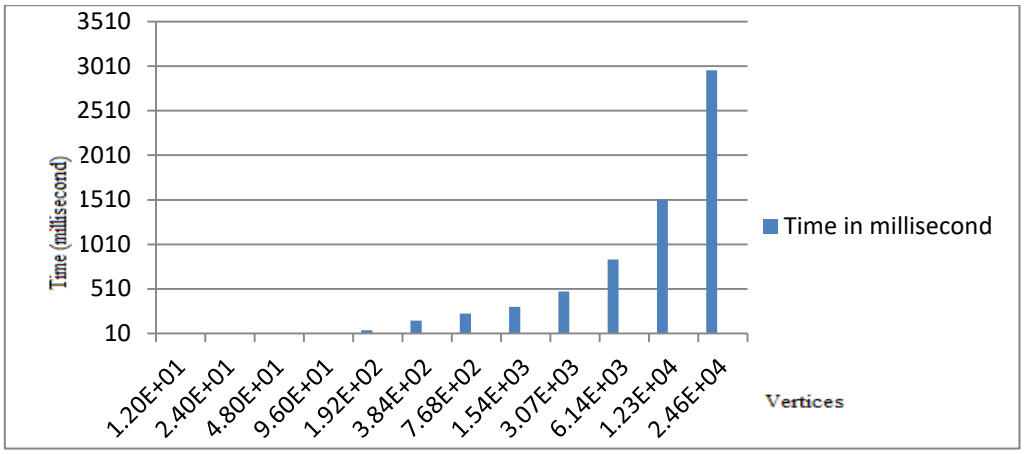

Figure 12: Represents table (7)

Table 8: GPU execution time in millisecond

Vertices	Time in millisecond
12	0.011872
24	0.011808
48	0.011776
96	0.012384
192	0.01254
384	0.012864
768	0.013664
1536	0.016768
3072	0.02624
6144	0.044672
12288	0.080384
24576	0.156224

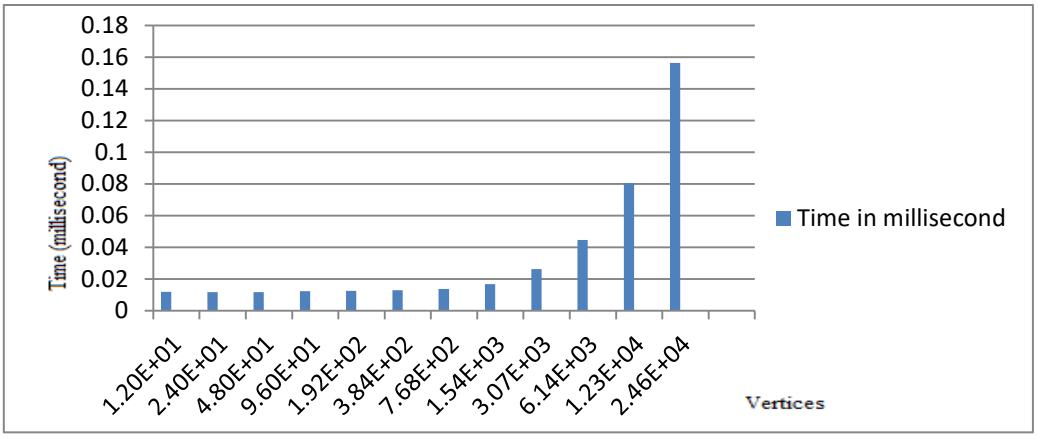

Figure 13: Represents table (8) GPU execution time in visual studio when all blocks and threads are exploited

5.7. Speed up

Speed up is the ratio between the sequential execution time to the parallel execution time.

$$Speed\ up = \frac{Sequential\ execution\ time}{Parallel\ execution\ time} \qquad (5)$$

5.7.1. MATLAB speed up

Speed up has been represented in figure (14) is taken from the ratio between table (6) and figure (11).

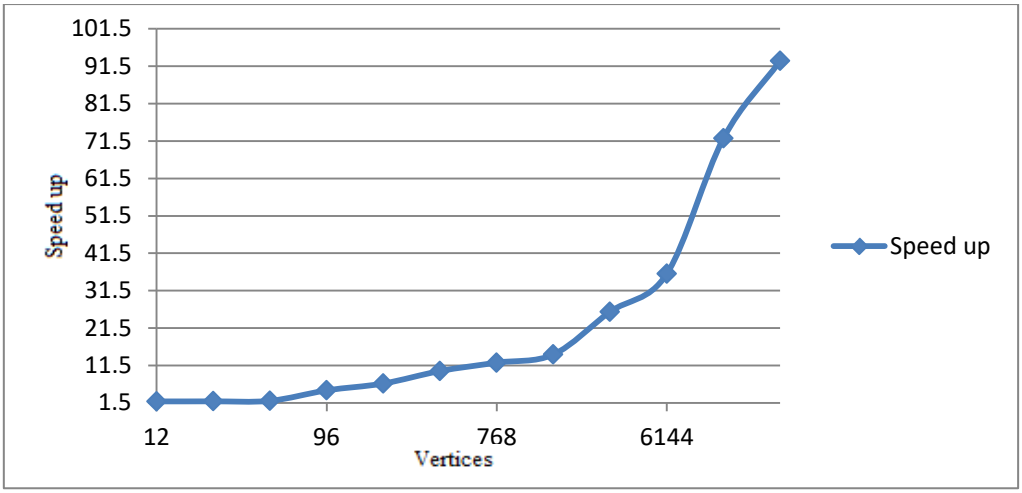

Figure 14: Speed-up of GPU in comparison to CPU using MATLAB

5.7.2. Visual Studio speed up

Speed up has been represented in figure (15). It is calculated from the ratio of table (7) to table (8).

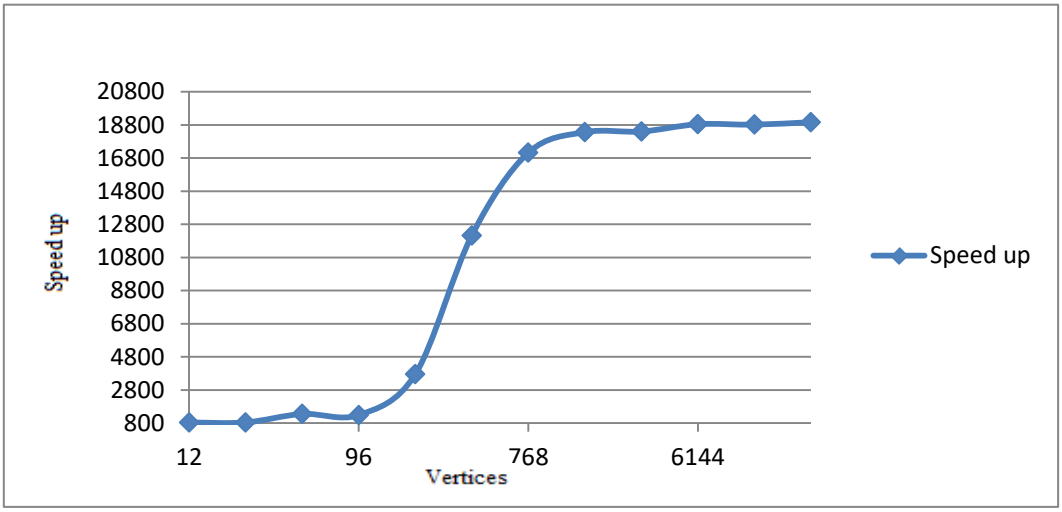

Figure 15: Speed-up of GPU in comparison to CPU using Visual Studio

Note that the CPU has been used in all the results above is Core i3, three cores is used together to perform sequential execution of vertices rotation.

5.8. Comparison between CPU single core and multicore using MATLAB.

As shown table (9), when a single core processor is used for a small set of vertices, the execution time is smaller than using multicore for the same set of vertices, because the time required to initializes cores and the communications among cores dominants over the benefits acquired from the parallelization when dealing with small set of vertices. And when increasing the number of vertices the multicore processor gives smaller execution time than single core. And the speed-up represents the performance of each of them. (See figure (16)).

Table 9: CPU single core VS CPU three cores and speed-up

Vertices	Time in sec (Single Core)	Time in sec (Three Core)	Speed-up
10	0.0044	0.0112	0.357142
100	0.0055	0.0127	0.43307
1000	0.0077	0.0394	0.177664
10000	0.1227	0.2284	0.53721
100000	8.313	1.0518	7.9035
1000000	77.809	8.8356	8.80630
10000000	839.5970	68.0995	12.3289

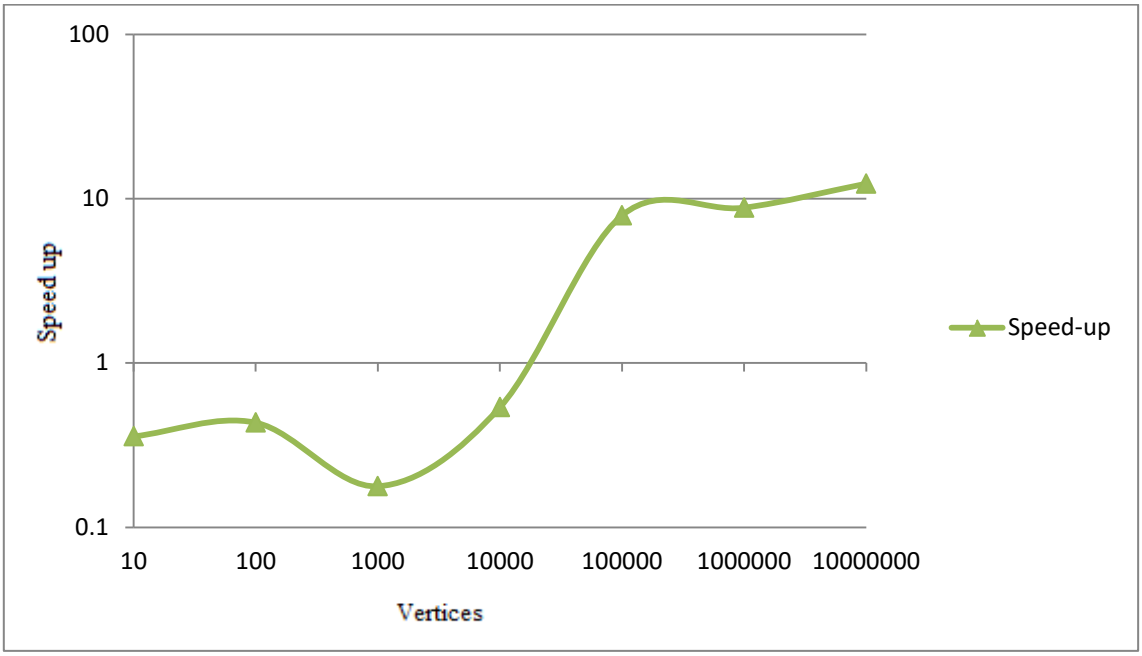

Figure 16: Represents speed-up between Single Core Processor and Multicore Processor

6. Discussion

Both MATLAB and Visual Studio have been used in this paper. Serial execution for vertices rotation has been achieved by using CPU, parallel execution for vertices rotation has been achieved by using GPU. From the results it can concluded that the Visual Studio give better result than MATLAB in terms of speed, GPU execution time using visual studio is lower than GPU execution time using MATLAB because access time to GPU's memory is different from one program to another. CPU Core i3 is used to compare between execution times require to rotate a set of vertices using three cores and single core. CUDA is a programing language that has been used and NVIDIA CUDA 7.5 is used for GPU driving.

7. Conclusion

One of the important notes is the speed of execution. This speed has been measured using different sets of vertices in terms of the time taken for translation and rotation. The execution acceleration is the most important feature of real time graphic applications. In this paper a general way for image rotation is achieved by using MATLAB and Visual Studio. From the results that have been discussed previously, MATLAB consumes more time than the Visual Studio to perform the same task in both serial and parallel execution. Figure (5) for MATLAB and figure (7) for Visual Studio explain that when the number of vertices exceeds the capacity, in terms of the blocks and threads, of the GPU. The execution time begins to increase exponentially with increasing the vertices. The aim of figure (5) and figure (7) is to find the size of GPU grid in terms of the numbers of blocks and threads, these figures show increasing of execution time when exceeding its capacity. The testing of GPU capacity is begin from Tile=1 (only one vertex in each block) until Tile=1000 (1000 vertex in each block). The results represent that the number of blocks equal to 100 which is close to the number of GPU cores, and the number of threads is in the range between (100, 500). Figure (6) for MATLAB and figure (8) for Visual Studio explains the actual number of blocks and threads using another set of vertices which has been chosen depending on partitioning the GPU to 96 block and changing Tiles number to find the actual number of threads which is concluded to be 256 threads. Figure (10) for MATLAB and figure (12) for Visual Studio present serial execution time for CPU. Visual Studio is faster than MATLAB in serial and parallel executions. Figure (14) for MATLAB and figure (15) for Visual Studio represent the speed up of GPU compared with CPU, the figures explain that Visual Studio is better than MATLAB to exploit the GPU to perform matrix multiplication and image rotation using parallel techniques. And concludes from these results that the single core processor is faster than the multicore processor in small set of vertices, and the multicore processor is faster than the single core for large number of vertices due to the time required to communicate and distribute data among processors when using small set of vertices.

References

[1] Sahin, Ibrahim. "A 32-bit floating-point module design for 3D graphic transformations". Scientific Research and Essays Journal, Vol.5, pp 3070-3081, 18 October 2010.

[2] Salomon, David. "Computer graphics and geometric modeling". Computers and Mathematics with Application Journal, Vol.38, pp. 289-298, 1999.

[3] Paeth, Alan W. "A fast algorithm for general raster rotation". Canadian Information Processing Society, 1986, pp. 77-81.

[4] Huang, Beilei. Edmund M-K. Lai, and A. P. Vinod. "Image resizing and rotation based on the consistent resampling theory". International Symposium on Intelligent Signal Processing and Communications Systems, Fep 2009, pp 1-4.

[5] Nickolls, John and Buck, Ian and Garland, Michael and Skadron, Kevin. "Scalable parallel programming with CUDA". Queue Journal. Vol.6, pp.40-53, 1/ March 2008.

[6] Karas, Pavel. "Gpu acceleration of image processing algorithms". PhD, Masarykova univerzita, Fakulta informatiky, 2011.

[7] Liu, Zhi Yuan, and Xue Zhang Zhao. "Research and Implementation of Image Rotation Based on

CUDA" In Advanced Materials Research Organization, Vol.216, Yuhang Yang, Xilong Qu, Yiping Luo and Aimin Yang, Ed. switzerland: Trans Tech Publications, 2011, pp. 708-712.

[8] Minhas, Umar Ibrahim, Samuel Bayliss, and George A. Constantinides. "GPU vs FPGA: A comparative analysis for non-standard precision" In Reconfigurable Computing: Architectures, Tools, and Applications, 10th, Vol.8405, Diana Goehringer, Marco Domenico Santambrogio, João M.P. Cardoso, Koen Bertels,Ed. Portugal, Springer International Publishing, 2014, pp. 298-305.

[9] Hochberg, Robert. "Matrix Multiplication with CUDA-a basic introduction to the CUDA programming model." Internet": http://www.shodor.org/media/content/petascale/materials/UPModules /matrixMultiplication/moduleDocument.pdf, [August 11 2012].

[10] Jang, Byunghyun, "Evaluation and enhancement of memory efficiency targeting general-purpose computations on scalable data-parallel GPU architectures." PhD, Northeastern University Department of Electrical and Computer Engineering, United States–Massachusetts, ProQuest Dissertations Publishing , 2010.

[11] Neelap, Akash Kiran. "Performance analysis of GPGPU and CPU on AES Encryption." Advanced level (degree of Master (Two Years)), School of Electrical Engineering Blekinge Institute of Technology, Karlskrona Sweden, 2014.

[12] Ali, Fakhrulddin H. Dawod, Amar I. "FPGA Based Implementation Of Concatenation Matrix". Al-Rafidain Engineering Journal, Vol. 18, pp. 15-31, 21/ June 2009.

[13] Ali, Fakhrulddin H. "Transformation Matrix for 3D computer Graphics Based on FPGA". Al-Rafadain Engineering Journal, Vol. 20, pp. 1-15, 1/Oct 2012.

[14] Olukotun, Kunle, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. "The case for a single-chip multiprocessor". The ACM Seventh International Conference on Architectural Support for Programming Languages and Operating Systems. 1996, pp. 2-11.

[15] R. Bittner, and E. Ruf, "Direct GPU/FPGA Communication via PCI Express". 2012 41st International Conference on Parallel Processing Workshops, on IEEE, Sep 2012, pp.135-139.

[16] M. Hulkkonen,"Graphics Processing Unit Utilization in CircuitSimulation", PhD diss, school of electrical engineering, Aalto University, 2011.

[17] Sodan, A.C., Machina, J., Deshmeh, A. Macnaughton, K., Esbaugh, B. "Parallelism via Multithreaded and Multicore CPUs". Computer Journal, pp. 24-32, March 2010.