Hardware/software co-design for a parallel three-dimensional bresenham's algorithm

Sarmad Ismae[1], Omar Tareq[2], Yahya Taher Qassim[3]
[1]Electronic Engineering Department, College of Electronic Engineering, Ninevah University, Iraq
[2]Computer Engineering Department, College of Engineering, University of Mosul, Iraq
[3]School of Electronic Engineering, Griffith University, Australia

Article Info

Article history:

Received Apr 8, 2018
Revised Jun 19, 2018
Accepted Jul 2, 2018

Keywords:

Bresenham's algorithm
FPGA
Line segmentation
Parallelization
System on Chip

ABSTRACT

Line plotting is the one of the basic operations in the scan conversion. Bresenham's line drawing algorithm is an efficient and high popular algorithm utilized for this purpose. This algorithm starts from one end-point of the line to the other end-point by calculating one point at each step. As a result, the calculation time for all the points depends on the length of the line thereby the number of the total points presented. In this paper, we developed an approach to speed up the Bresenham algorithm by partitioning each line into number of segments, find the points belong to those segments and drawing them simultaneously to formulate the main line. As a result, the higher number of segments generated, the faster the points are calculated. By employing 32 cores in the Field Programmable Gate Array, a line of length 992 points is formulated in 0.31µs only. The complete system is implemented using Zybo board that contains the Xilinx Zynq-7000 chip (Z-7010).

Corresponding Author:

Yahya Taher Qassim,
School of Electronic Engineering,
Griffith University,
170 Kessels Road, Nathan, QLD 4111, Australia.
Email: yahya-taher.qasim@griffithuni.edu.au

1. INTRODUCTION

One of the essential and main activities in computer graphics is the straight line drawing. To get the lines displayed in a short time, the computation speed of the applied algorithm is crucial [1]. Several algorithms have been developed and employed for drawing straight lines [2], [3]. However, the utilization of complex and inefficient algorithms in generating straight lines then displaying their pixels could be slow and unacceptable to the user [4]. Such slow algorithms may include heavy computations that consume time and power. Bresnham algorithm was highly regarded among other equivalent algorithms due to its suitability and efficiency since floating-point, multiplication and division operations do not exist in addition to its numerical scaling of integers only [5]. There are several works in the literature concerning line rasterization including Bresenham algorithm, which had been implemented in different schemes and methods. In [6], an improvement to Bresenham algorithm was performed by determining the number of pixels for each line then filling those pixels. The modified algorithm of [7], which combines the symmetry nature of lines with Bresenham algorithm, improved the drawing speed by fifty percent; however, the work was only experienced in simulation.

Drawing long straight lines negatively affects the efficiency of Bresenham algorithm, where the rasterization speed is extremely reduced in addition to missing the pixels of correlation [6], [7]. In order to speed up the drawing algorithm, hardware utilization is necessary. A popular hardware platform utilized for digital system design is the field programmable gate array (FPGA) [8]. However, when the length of the line is extended, the time required to calculate the extra points would increase as well since Bresenham algorithm is based on finding the points of the specified line sequentially. Therefore, even with the involvement of hardware, the delay in drawing lengthy lines is likely to happen.

In this paper, we have solved the problem of drawing lengthy straight lines in an extremely short time. Our approach of line segmentation and parallelization in implementing the 3D-Bresenham algorithm on single SoC (System on Chip) [9], [10] is presented. The parallelization approach is to handle several procedures at the same time and execute them independently [11]. Therefore, we have segmented the line into several shorter lines of equal length and considered every segment as a separate process. In such a case, the time consumed in implementing one procedure is the same time in implementing multiple procedures. As a result, any extension in the length of the line can be covered by adding an additional procedure to compute the points reflecting that increase and this would not add further running time.

This article is organized as follows; in Section 2, a brief description on the 3D Bresenham algorithm is outlined. Section 3 explains our line segmentation and parallelization approach utilized for drawing straight lines followed by the Zynq implementation in Section 4. In Section 5, the obtained results and analysis are given whereas the conclusions and future work are included in Section 6.

2. THREE DIMENSIONAL BRESENHAM'S LINE ALGORITHM

The flow chart for three dimensional Bresnham's algorithm [12] is presented in Figure 1, where the pixels of the line segment are generated in a three dimensional space. For each pixel, the x, y, z coordinates are calculated in the object space. Bresnham's algorithm starts from one end-point of the line to the other end-point by calculating one point at each step. As a result, the calculation time for all the points depend on the length of the line thereby the number of the total points presented [13].

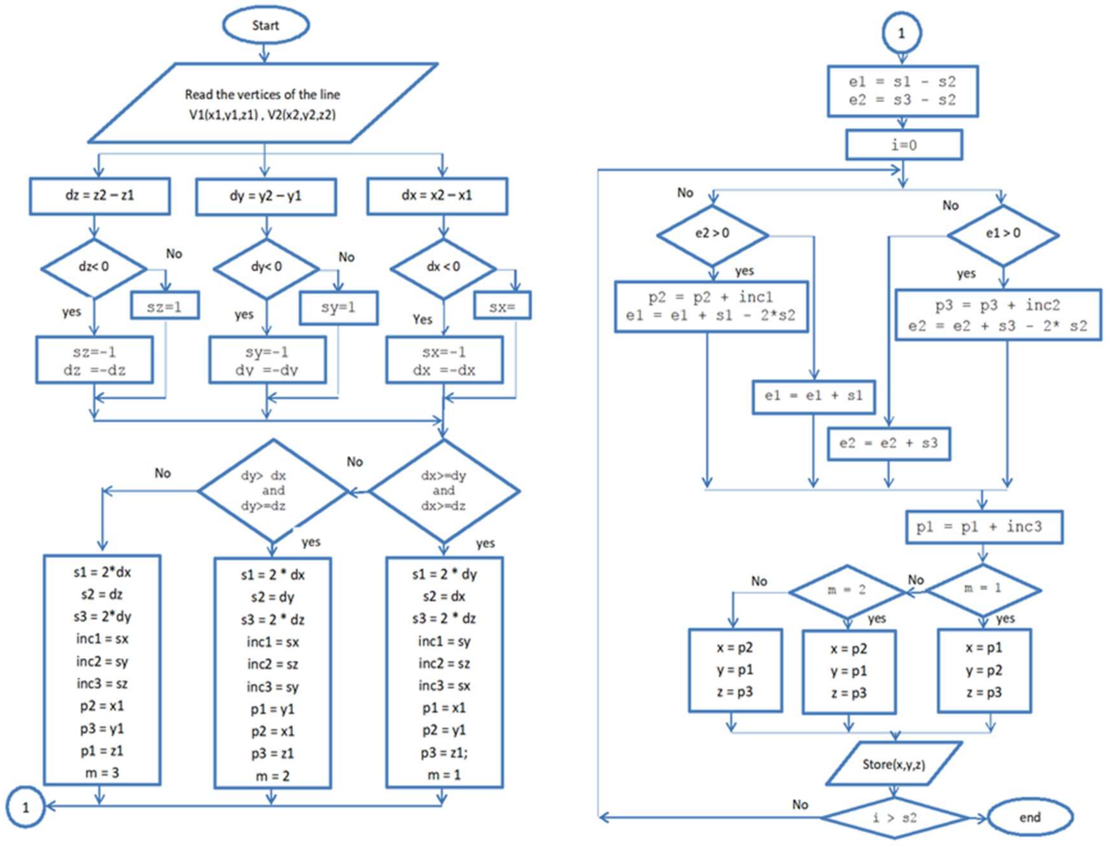


Figure 1. The flow chart for the 3D-Bresenham algorithm

From Figure 1, the vertices of the line segment (x_a, y_a, z_a) and (x_b, y_b, z_b) are read first then the greatest coordinate difference (*dx, dy, dz*) is calculated. The variables *s1, s2, s3, inc1, inc2, inc3, p1, p2, p3* and *m* are given their assigned values according to the greatest coordinate difference. Following to this, the error values (*e1* & *e2*) are computed in order to find the increment value of *x, y, z.* The next step is to find the intermediate pixels of the line segment. Those points can be calculated each time by adding the incremented value to the *x, y, and z* coordinate. Finally, store the calculated point and return to calculate the new increment value. When the coordinate difference is equal to zero, the algorithm ends [12].

3. MATERIAL AND METHODS

As previously mentioned, in Bresenham algorithm, the line pixels are generated one by one beginning at the start point a (xa, ya, za) towards the end point b (xb, yb, zb). In such a case, the time required to compute all the points is increasing as long as the line becomes longer, leading to a slower plotting process. Our approach in implementing the 3D-Bresenham algorithm is performed by line segmentation and procedure parallelization. The line is divided into equal-length segments using the ARM A9 Cortex or the processor system (PS) available on the Zynq chip followed by sending them to the FPGA located on the same chip. The FPGA or the programmable logic (PL) contains up to 32 individual cores and each of those cores can perform a separate procedure concurrently with other cores. Therefore, to draw a line, the latter can be divided into up to 32 equal-length segments then calculate the points of each segment separately in one of the cores and in parallel with the other segments. As a result, the number of cores employed divides the time required to find the total points of the line. In other words, Bresenham algorithm is implemented simultaneously with a number of times equivalent to the number of cores engaged in the process.

The time required to draw a line can be controlled. For example, the increase of segments' number decreases the total time required to draw the line or keep the time constant when the length of the original line increased by generating additional segments. Figure 2 describes the Zynq architecture [14] showing the PS part and only two cores (out of 32) at the PL side. The AXI4-Lite is the communication bus interface between the ARM cortex A9 processor and the FPGA. It can be programmed to work in more than one protocol. We have appointed it as AXI4-lite bus, which is simple, easy and does not require memory mapping [15].

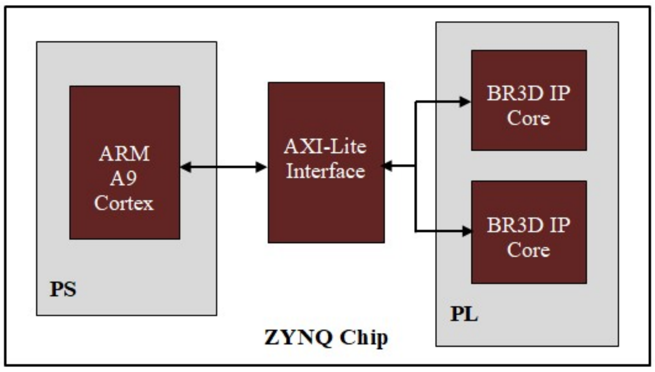


Figure 2. The Xilinx Zynq SoC including the ARM cortex (PS) and showing two of the FPGA (PL) cores. The BR3D refers to Bresenham algorithm (three-dimensional)

For the practical implementation of Bresenham algorithm, the line pixels are described by the coordinates (x, y, z). Each pixel is represented by 32 bit in total as follows: 10 bit for x, 10 bit for y, 10 bit for z coordinate and 2 bit are not applicable. Since the FPGA-BRAM (connected to each core) is of size 1024×32 bit, which is fixed for each line segment, the straight line plotting could start at the coordinate (0, 0, 0) and ending at the coordinate (1023, 1023, 1023). The maximum length of a single line segment is 1024 point. As mentioned, the number of cores employed depends on the number of segments where each core is performing pixel computation of one segment at a time. Table 1 shows different number of cores employed or number of line segments and the maximum length of rasterized line in pixels. When all the FPGA cores are involved in line computation, the maximum length of the line generated is 32768 pixels. The number of cores/segments versus maximum length of line in pixels shown in Table 1.

Table 1. The Number of Cores/Segments versus Maximum Length of Line in Pixels

No. of cores or line segments	Max length of rasterized line (in pixel)
2	1024 * 2 = 2048 (2K pixel)
4	1024 * 4 = 4096 (4K pixel)
8	1024 * 8 = 8192 (8K pixel)
16	1024 * 16 =16384 (16K pixel)
32	1024 * 32 =32768 (32K pixel)

4. THE ZYNQ IMPLEMENTATION

As mentioned, the Zynq chip located on the Zybo board contains two main computation parts: the ARM processor (PS) and the FPGA (PL) as in Figure 2. The PS is allocated for algorithms with high computation complexity whereas the PL is utilized to implement logical system design [16]. Therefore, the whole system is programmed in C language to perform the following tasks for each of the PS and the PL parts. The PS is directed to perform line segmentation by partitioning the main line into number of segments assigned according to the number of PL cores. In other words, the processor will generate the start and end coordinates, i.e. the (x, y, z) coordinate for the terminal points for every segment before sending them to the FPGA. Since the terminal points of each segment are known, the FPGA cores start calculating the total segment's points according to the 3D Bresenham algorithm. At the end, all the cores that involved in computation complete at the same time and the coordinates of all points are calculated and stored in the BRAM. When all the points are computed and stored, their values are send back to the PS part one by one for verification. Finally, the points are rasterized on the computer screen formulating the specified line.

4.1. Core operation

Figure 3 shows the internal architecture of one of the FPGA 32 cores. It consists of numbers of 32-bit register utilized for initialization and signal separation. The coordinates of the terminal points P1 and P2 are entered through Reg0 and Reg1 respectively. The three dimensional Bresenham algorithm (BR3D) unit contains the logic configured reflecting this algorithm, which computes the points of the assigned segment. When the unit completes calculating the points, bit10, the Ready (Rdy) bit in Reg2 is set and bit0 to bit9 will hold the number of the calculated points (NCP). Each calculated point is stored in the dual port block RAM that can be fetched through Reg4 (P_out) when the corresponding address is given through Reg3. Figure 4 illustrates the block diagram designed to implement Bresenham algorithm inside the BR3D unit in Figure 3. In every clock pulse, a new point is generated from point 1 (x1, y1, z1) and point 2 (x2, y2, z2). The architecture contains three subtraction units, three units for the absolute value (ABS) and three comparators, a multiplexer and finally the calculation unit. The intermediate signals s1, s2, s3, inc1, inc2, inc3, p1, p2, p3 and p4 were described in Figure 1. The combination of all of these blocks execute Bresenham algorithm.

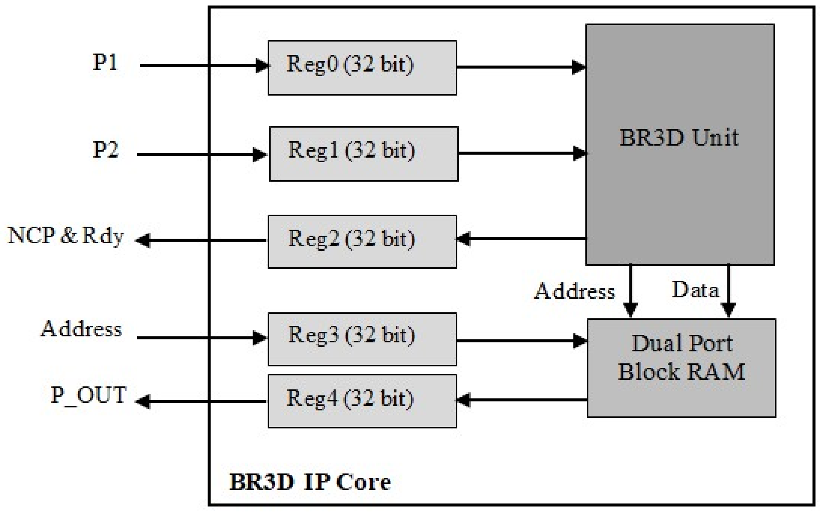


Figure 3. The internal architecture of the FPGA core designed for the 3D Bresenham algorithm

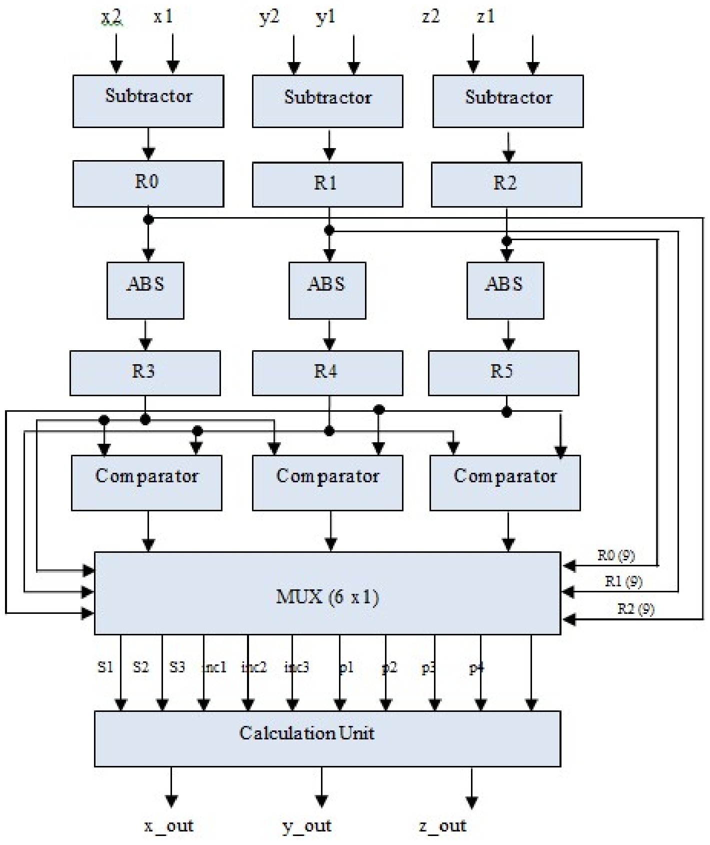


Figure 4. The implementation of Bresenham algorithm inside the FPGA cores

In Figure 5, the timing simulation of the main signals in Figure 4 is shown. The start point (x1, y1, z1) and the end (x2, y2, z2) coordinates are all inputs to the architecture giving the calculated output point (x_out, y_out, z_out) varies at each clock pulse. The output points combined represent the line segment. The start point in this example is (3, 2, 5) and the end point is (12, 15, 20). It can be noticed from Figure 5. that some coordinates may change value every two-clock pulses such as z-out whereas one clock pulse is enough to change value in case of the x-out coordinatedue due to the variation in the coordinate difference between its start and end.

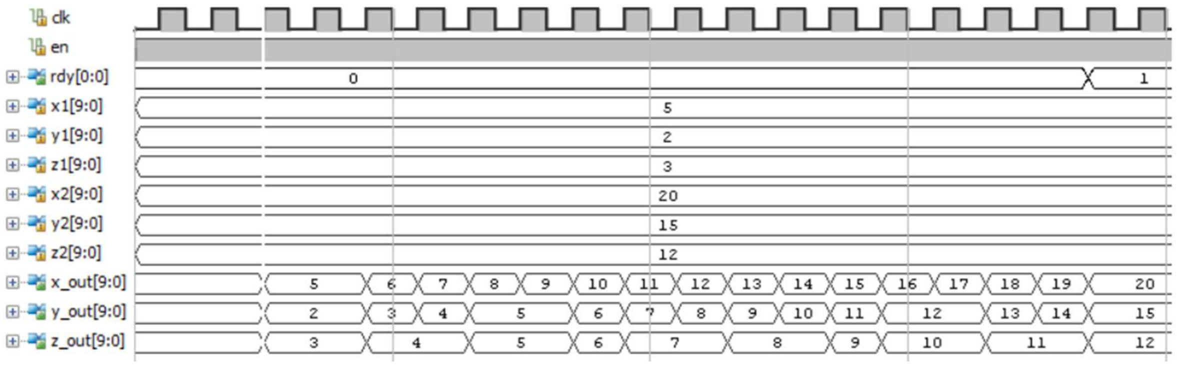


Figure 5. The timing waveform for the input/ output coordinates of one segment calculated using 3D-Bresenham algorithm

4.2. Timing constraints

Meeting timing constraints remains an essential part in any digital system design. In this work, all the design procedures are performed through the Vivado software package including design optimization, which is essential to meet timing requirements. For example, when a specified frequency of operation is assigned, such as 100MHz, Vivado will optimize the design according to the period of "create clock". This is essential to achieve the targeted frequency according to the worst failing path in the design. The Xilinx recent release of Vivado Design Suit 2017.2 supports the Zynq SoC with a wide variety of FPGA devices. It supersedes previous design tool by its additional features of high-level synthesis and SoC [17].

5. RESULT AND DISCUSSIONS

5.1. Timing analysis

In our design, the Zybo board operates at 100MHz clock rate and the architecture calculates one point in every segment in one clock pulse. The number of FPGA cores employed share calculating the number of line points. The hardware runtime is reduced to half when the number of cores employed is doubled. The fastest runtime achieved is 0.31µs when all the 32 FPGA cores are involved. This time represent the segmentation time (in the PS) in addition to the time required for points' calculation (in the PL). Table 2 lists a comparison between this work and other relevant works. Although the comparison is not on exact coordinates, the runtime in this work clearly advances the running time of other works. Figure 6 reflects the decrease in the hardware running time against the increase in the number of cores employed for the line of 992 point.

Table 2. Runtime Line Drawing Compared to the Literature

Reference	Year	Start coordinates	End coordinates	Operating environment	Running time
This work (32 FPGA cores)	2018	(0, 0, 0)	(992, 992, 992)	Zybo SoC board	0.31µs
This work (one FPGA core)	2018	(0, 0, 0)	(992, 992, 992)	Zybo SoC board	9.92µs
[18] Bresenham 3D	2013	(0, 0, 0)	(1000, 1000, 1000)	Spartan 3E FPGA	13.16µs
[18] modified (less hardware) Bresenham 3D	2013	(0, 0, 0)	(1000, 1000, 1000)	Spartan 3E FPGA	14.71µs
[6] line generation based on line pixels	2011	(0, 0)	(1000, 1000)	Personal computer	1.25ms
[19] Bresenham improved algorithm	2010	(0, 0)	(1000, 1000)	Personal computer	1.36ms
[20] A fast line rasterization algorithm	2008	(0, 0)	(1000, 1000)	Personal computer	1.72ms

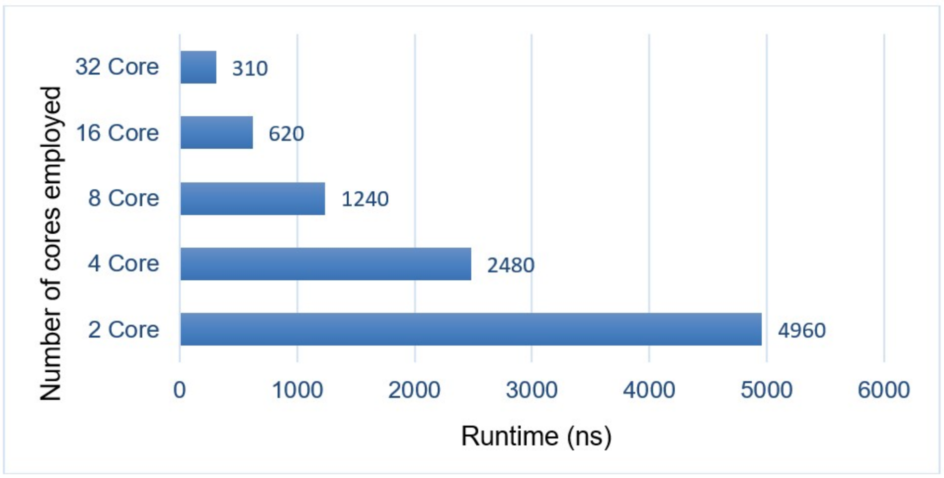
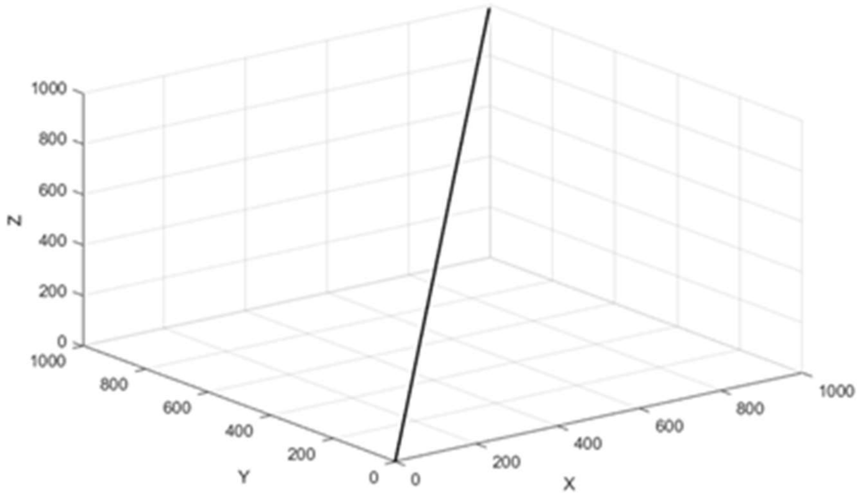


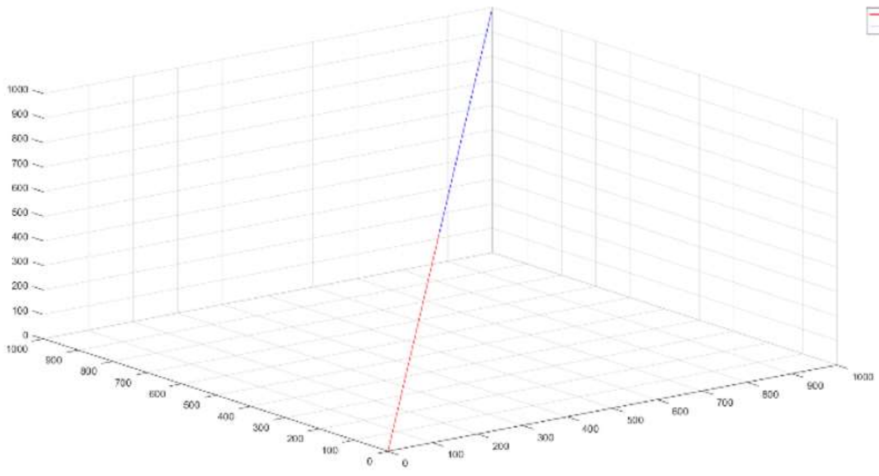
Figure 6. The reciprocal relationship between the FPGA cores employed and the hardware runtime

5.2. Graphical analysis

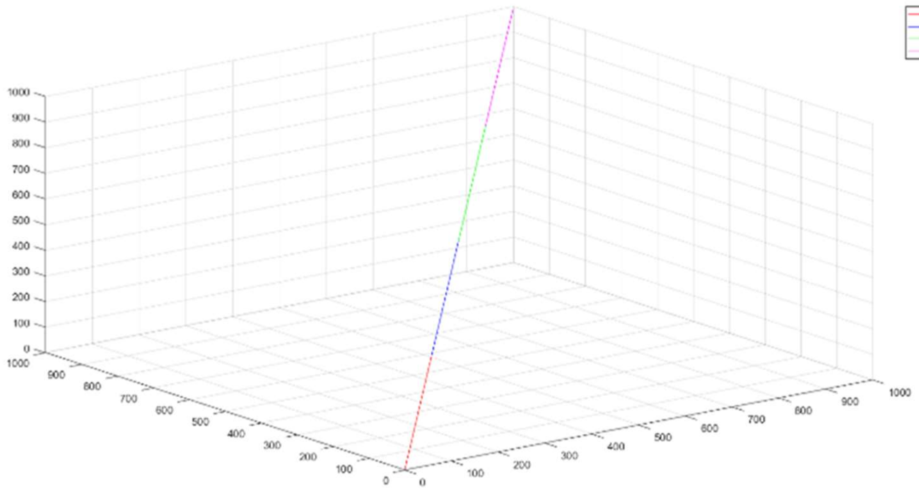
The main 3D line of 992 point is depicted in Figure 7 in four different cases. The number of segments generated from the main line using our approach of parallel implementation of Bresenham algorithm depends on and equal to the number of FPGA cores utilized. The generated points are plotted using Matlab. Colors are used to distinguish the start and end of each segment.



(a)

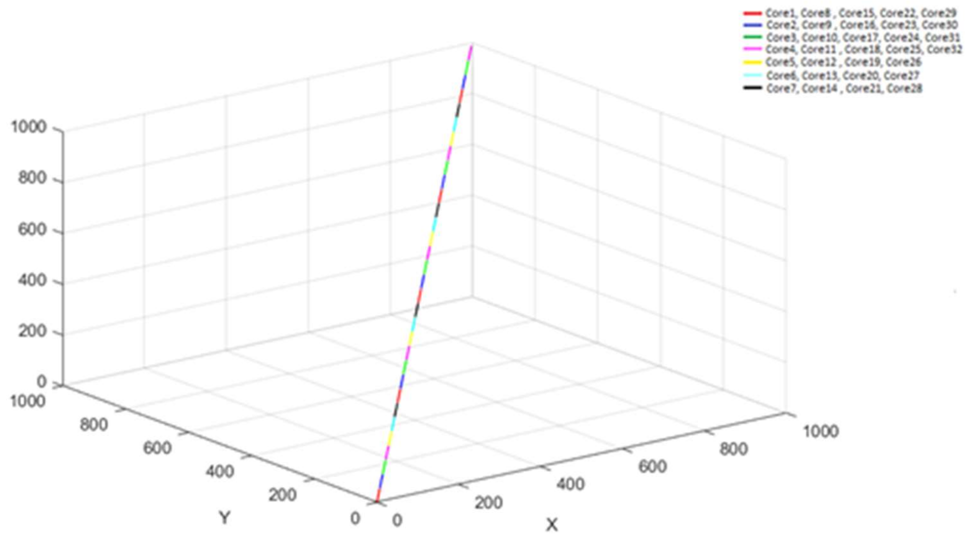


(b)



(c)

Figure 7. The line of 992 point computed in (a) one-FPGA core, (b) 2-FPGA cores, (c) 4-FPGA cores, (d) 32-FPGA cores and formulated with 32 different color segments each of 31 point



(d)

Figure 7. The line of 992 point computed in (a) one-FPGA core, (b) 2-FPGA cores, (c) 4-FPGA cores, (d) 32-FPGA cores and formulated with 32 different color segments each of 31 point (*continue*)

5.3. Resource evaluation

Each of the FPGA cores in the Zynq SoC contains variety of logic resources essential to build the assigned digital circuit by the user, such as the look-up tables (LUT), block RAM and flip-flops. The more FPGA cores are employed, the more utilization of resources is resultant. The parallel usage of these identical cores leads to a very fast implementation of Bresenham algorithm (0.31μs) for a line with a high number of points (992 point). The high ability of the Zybo platform and the Vivado software package lead to excellent achievement in the running time of Bresenham algorithm. However, the percentages of resource utilization are increased directly with the higher number of cores employed as Figure 8 shows.

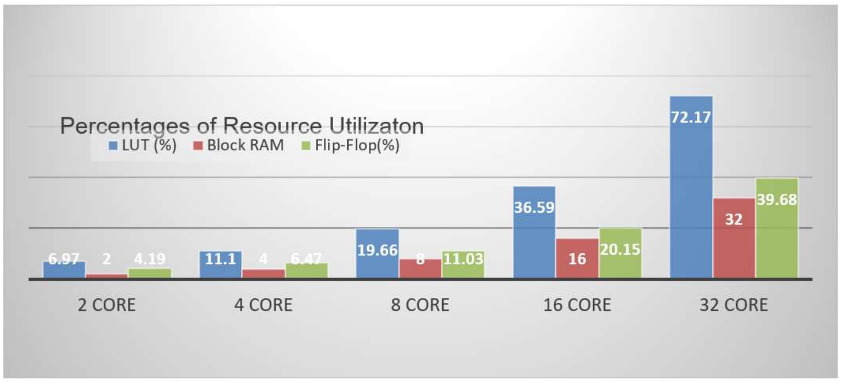


Figure 8. The increase in the number of FPGA cores employed against the increase in the percentages of resource utilization

6. CONCLUSION

The characteristic of procedure parallelization is extremely practical for any digital system design when the architecture of the available hardware supports parallelism. With the employment of the Zynq SoC that include 32 identical cores in its PL part, parallel processing of identical procedures was achievable. In this paper, this was applied to divide a line of 992 points into a maximum of 32 identical segments and implement Bresenham algorithm in parallel to compute the points of each of those segments. The hardware runtime for a line of 992 point was reduced from 9920ns (in case of normal implementation of Bresenham algorithm on the Zybo board) down to only 0.31µs when the parallel implementation is handled. This makes the followed procedure suitable for real time graphical applications.

Future applications to the concept of parallelization can include drawing a cube or other complex shapes. It can also be effective in drawing a 3D polygon shape using the same architecture in calculating one polygon line per core to end with formulating all the polygon lines at the same time, or even rendering a polygon mesh. In addition, drawing several lines in parallel (maximum 32 lines) will speed up the overall graphical presentation. Moreover, the hardware-software co-design can also be expanded to facilitate the association between the PL (FPGA) and the PS (ARM-Cortex) blocks in the Zynq chip.

REFERENCES

[1] X. Liu and K. Cheng, "Three-dimensional Extension of Bresenham's Algorithm And Its Application In Straight-Line Interpolation," *Part B: J Engineering Manufacture*, vol. 216, pp. 459-463, 2002

[2] H. Mei-gui, *et al.*, "Based on Parallel Filling [J]," *Jinan University Journal (Natural Science Edition)*. vol. 18(3), pp. 212-214, 2004.

[3] K. OuYang, *et. al*, "Eight-Step Linear-Generation Algorithm Based on Symmetry [J]," *Computer Science*, vol. 35(3), pp. 247-250, 2008.

[4] A. T. M. Shafiqul Khalid and M. Kaykobad, "An Efficient Line Algorithm," in *the 39th Midwest Symposium on Circuits and Systems*, 1996, pp. 1280-1282.

[5] J. Shen, *et al.*, "An Improved Line-Drawing Algorithm for Arbitrary Fractional Frequency Divider/Multiplier Based on FPGA," *Journal of Engineering Science and Technology Review*, vol. 6(5), pp. 90-94, 2013.

[6] L. Yan-cui, *et al.*, " A Straight Line Generation Algorithm based on Line Pixels," in *the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, 2011, pp. 466-469.

[7] L. Zheng, *et al.*, " Modified Line Algorithm based on ORGFX," in *the 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2014, pp. 42-45.

[8] B. Kanigoro, *et al.,* "Overview of Custom Microcontroller Using Xilinx Zynq XC7Z020 FPGA," *TELKOMNIKA Indonesian Journal of Electrical Engineering,* vol. 13(1), pp. 364-372, 2015.

[9] T. Adiono, *et al.*, "An SoC Architecture For Real-Time Noise Cancellation System Using Variable Speech PDF Method," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 5(6), pp.1336-1346, 2015.

[10] H. Yang, *et. al.,* "An Embedded Auto-Leveling System based on ARM and FPGA," *TEKOMNIKA Indonesian Journal of Electrical Engineering,* vol. 11(12), p. 7094-7101, 2013.

[11] W. E. Wright, "Parallelization of Bresenham's Line and Circle Algorithms," *IEEE Computer Graphics and Aplications*, vol. 10(5), pp. 60-67, 1990.

[12] J. E. Bresenham, "Algorithm for Computer Contro of a Digital Plotter," *IBM Systems Journal,* vol. 4(1), pp. 25-30, 1965.

[13] C. Au and T. Woo, "Three Dimensional Extension of Bresenham's Algorithm with Voronoi Diagram," *Computer-Aided Design,* vol. 43, pp. 417-426, 2011.

[14] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (accessed on 11 Jan 2018).

[15] L. Crockett, *et al.*, "The Zynq Book Tutorials", 2015. Available on http://www.zynqbook.com/download-tuts.html

[16] https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (accessed on 25 Jan 2018).

[17] https://www.xilinx.com/products/design-tools/vivado.html (accessed on 5 Jan 2018).

[18] B. Younis and N. Sheet, "Hardware Implementation of 3D-Bresenham's Algorithm using FPGA," *Tikrit Journal of Engineering Sciences*, vol. 20(2), pp. 37-47, 2013.

[19] Y. Jia, *et al.*, "A Modified Bresenham Algorithm of Line Drawing [J]," *China Journal of Image and Graphics,* vol. 13(1), pp. 158-161, 2008.

[20] L. Niu and Z. Shao, "A Fast Line Rasterization Algorithm based on Pattern Decomposition [J]," *Journal of Computer Aided Design & Computer Graphics*, vol. 22(8), pp. 1286-1292, 2010.