Performance Characterization of Rasterization Algorithms for Reconfigurable Graphics Processor

Junyong Deng* , Qingqing Ma† and Zekun Ye‡
*Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: djy@xupt.edu.cn Tel: +86-29-18192964908
†Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: 784597682@qq.com Tel: +86-29-15709106471
‡Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: 1226785825@qq.com Tel: +86-29-15771783882

***Abstract*— For different rasterized scenes, even different areas of the same scene, the performance bottleneck of rasterization may be different, and current graphics processors cannot choose the appropriate rasterization algorithm according to the specific rendering scene. Therefore, a reconfigurable graphics processor that supports switching of algorithms to achieve the best performance is a promising choice. The existing graphics processor suffers the constraints of calculation, memory and power consumption in different rasterization application scenarios. Therefore, it is very important to determine how to schedule different rasterization algorithms with different performance according to the actual requirements in the reconfigurable graphics hardware. This paper evaluates and analyzes the performance characteristics of three main-stream rasterization algorithms (scan-line filling algorithm, edge filling algorithm, and flood filling algorithm) in different application scenarios. Pearson correlation coefficient (PCC) analysis is leveraged to analyze the relationship between performance/energy and evaluation metrics. Based on these performance characterization data, this paper puts forward some reconstruction suggestions for the reconfigurable graphics processor. We hope to contribute to reconfigurable graphics processing.**

***Index Terms*— Rasterization Algorithms; Performance Characterization; Reconfigurable Graphics Processor**

I. INTRODUCTION

Rasterization is a very important part in the GPU (graphics processing unit). It is the process of converting basic primitives (points, line segments, triangles) into pixels. The rasterization quality and efficiency of primitives directly affect the performance of the entire GPU pipeline [1]. Among all the primitive objects of the GPU, triangle primitives are the most basic and most important primitives, and also the basic primitives that make up any other more complex 2D or 3D objects. The process of triangle rasterization includes a large number of arithmetic operations and complex logic control, which has become a key link for improving the performance of graphics processors [2].

The most important indicator of triangle rasterization is efficiency [3], that is, how many triangle primitives the GPU can process in a unit of time. In response to the increasing demand for GPU processing performance, high-performance GPUs usually integrate dozens to hundreds of parallel rasterization modules to improve performance. But relying solely on increasing the number of modules to improve the rasterization efficiency will increase the size and complexity of the chip and increase the design cost. The efficiency of a single rasterization module should be improved on the basis of expanding the number of rasterization modules [4]. There are many factors that affect the performance of rasterization. For different rasterized scenes, or even different areas of the same scene, the performance bottleneck may be different [5], and the current GPU does not make all algorithms adapt to real rendering scenes [6]. Therefore, a reconfigurable

graphics processor that supports the loading and switching of algorithms to achieve the best performance is a promising choice.

Rasterization algorithms are usually subject to computational constraints, memory constraints, and power consumption constraints in different application scenarios. Deciding how to autonomously schedule different rasterization algorithms based on actual needs is critical, but there is no suitable standard to decide when to start reconfiguring the graphics processor and which algorithm should be selected. Therefore, for the design of reconfigurable graphics processors, the first step is to establish a performance evaluation model, which can provide relevant data to decide when and how to perform reconstruction [7]. This paper focuses on three widely used rasterization algorithms: scan-line filling algorithm [8], edge filling algorithm [9], and flood filling algorithm [10], and conducts characterization analysis in different rendering scenarios. Through the comparison of characteristic data such as data movement, computation, power consumption, IPC and cache MPKI, some reconfiguration suggestions in the reconfigurable graphics processor are proposed.

The contributions of this paper can be summarized as follows:

- The performance of three commonly used rasterization algorithms: scan-line filling algorithm, edge filling algorithm and flood filling algorithm, including data movement per pixel, computation per pixel, power consumption per pixel, and the MPKI of each level of cache.
- In different graphics rendering scenarios, make recommendations for algorithm scheduling based on the comparison of performance data.

The rest of this paper is organized as follows: Section 2 summarizes the development trend of reconfigurable graphics processors, briefly introduces three popular rasterization algorithms, and the hardware platform on which this paper runs all analyses. Section 3 describes the graphical scene data set, analysis tools and indicators. In Section 4, we introduced the performance comparison of the three algorithms in detail, and provided suggestions for reconstruction. Section 5 summarizes this article.

II. BACKGROUND

A. *Reconfigurable Graphics Processor*

Reconfigurable computing is considered to be an effective solution that combines the high flexibility of traditional processors with the high processing efficiency of application-specific integrated circuits, and it has better performance among the key indicators of chips such as performance, power consumption, and functional flexibility. The balance will be the future development direction of computer architecture [11]. The fine-grained reconfigurable architecture offsets the high repetitive engineering costs of application specific integrated circuits through reconfigurable logic element arrays, and achieves higher energy efficiency by avoiding the overhead of general-purpose processors. However, the bit-level reconfigurability in fine-grained reconfiguration will generate a lot of area and power consumption overhead [12]. The coarse-grained reconfigurable architecture (CGRA) [13-14] has a denser reconstruction unit, which can significantly reduce the system configuration time and configuration memory consumption, and can also share the reconstruction overhead, but in programmability, flexibility and productivity are not yet mature [15-17]. Therefore, for graphics processors, reconfigurable computing is a trend. However, reconfiguration standards for graphics processors have not been established to determine when and how to refactor based on actual needs. This paper takes the rasterization algorithm in graphics rendering as an example, discusses the performance characteristics of different algorithms, and puts forward some reconstruction suggestions in reconfigurable graphics processors.

B. *Main-stream Rasterization Algorithms*

Scan-line filling algorithm: Using a horizontal scan-line to scan a polygon composed of multiple end-to-end line segments from top to bottom (or from bottom to top), and each scan-line generates a series of intersections with certain edges of the polygon. Sort these intersection points according to the x coordinate, and pair the sorted points in pairs, as the two end points of the line segment, and draw a horizontal straight line with the filled color. After the polygon is scanned, the color filling is completed, and it can

be summarized into the following four steps: find the intersection point (that is, the intersection point of the scan-line and the polygon); sort the intersection points; match the sorted points in pairs; update the scan-line to determine whether the polygon scan is completed.

Edge filling algorithm: complement each side to the right, you can process each side of the polygon in any order. When processing each side, firstly find the intersection of the side and the scan-line, and then complement all the pixels to the right of the intersection on each scan-line. After all the sides of the polygon are processed, the filling is completed.

Flood filling algorithm: starting from a starting node, extracting or filling the neighboring nodes with different colors, until all the nodes in the closed area have been processed. It is a classic algorithm that extracts several connected points from a region to distinguish them from other adjacent regions (or dye them in different colors). The algorithm uses three parameters: the start node, the target node characteristics, and the processing to be performed on the extracted object.

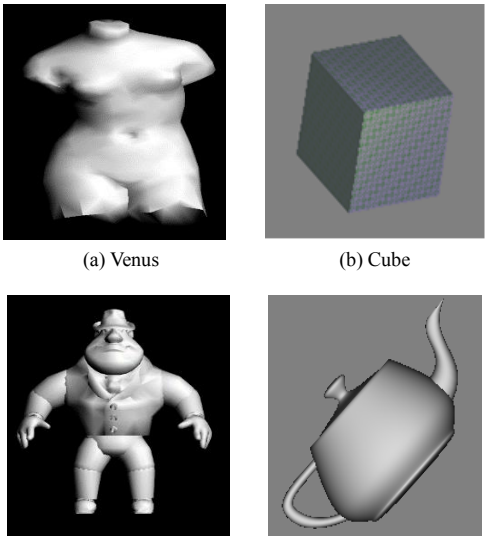
C. *Hardware Platform*

This paper is based on the Coffee Lake architecture of Intel's eighth-generation Core processor [18-19] to make statistics on the hardware performance of different rasterization algorithms. Coffee Lake is Intel's next-generation 14nm process processor after Skylake and Kaby Lake. It consists of 6 cores. Each core has a 32KB L1 data cache and 32KB L1 instruction cache, a 256KB L2 cache and 9MB L1 cache. Coffee Lake can decode and exit at most 5 instructions per cycle. In theory, the number of instructions executed per clock cycle is at most 5.

III. METHODOLOGY

A. *Graphical Scene Dataset*

In order to analyze the performance differences of algorithms in a variety of different rendering scenes, this paper selects four representative rasterized scenes, as shown in Figure 1. During the performance evaluation of graphical analysis, the data input has different data sizes and scenarios [20]. Due to the large number of graphics scenes, the number of vertices and triangles includes ranges from tens of thousands to hundreds of thousands, and the number of pixels per frame is about tens of thousands. In order to collect performance data reliably, Table I lists selected rendering scene data sets.



(a) Venus (b) Cube

(c) Cartoon (d) Teapot

Fig. 1 Graphics Scenes for Performance Characterization

Table I Rendering scene dataset

Dataset	#Vertices	#Triangles	#Pixels/frame
Venus	4,254	1,418	47,650
Cube	48	16	12,075
Cartoon	21,372	13,376	78,081
Teapot	767	256	20,186

B. *Analyzing Tool*

This paper uses the Intel VTune performance analyzer [21] to analyze the algorithm. VTune is a tool for analyzing and optimizing program performance. It can perform performance analysis on 32-bit and 64-bit x86 computers and assist in various code analysis, including stack sampling, thread analysis, and hardware event sampling. The result of the analyzer consists of some details, such as the time spent in each subroutine, which can go down to the instruction level. In order to calculate the IPC, data movement, computation, power consumption and cache MPKI of different algorithms in different rendering objects, this paper will collect the cycle, instruction count, load, store, branch, and miss count in each level of cache for each algorithm program through the

analyzer, and the corresponding performance indicators can be studied through these event sets.

C. Performance Metrics

The performance indicators for comparative analysis in this paper are: IPC, data movement, computation, power consumption, and L1, L2, and L3 data cache MPKI, which will be introduced below [22].

IPC: IPC is a basic performance indicator that represents the average number of instructions executed per clock cycle. For example, the theoretical best IPC is 5, but it is affected by various factors, such as long-latency memory, floating-point numbers, or single instruction multiple data (SIMD) operations, instructions that have not exited due to branch, insufficient front-end instructions, etc., reduced the observed IPC, resulting in the actual situation of IPC rarely reaching the ideal situation.

Data movement: the cost of data movement is much higher than the calculation operation, because the energy cost of external memory access is hundreds or even thousands of times of on-chip arithmetic or logical operations. Through statistical performance events, according to formula (1), this paper calculates the amount of data movement of each pixel of the three algorithms in multiple scenes. Among them, #load and #store are the total number of load and store instructions, used to indicate data movement, #pixels_per_frame is the number of pixels per frame.

$$data\_mov = \frac{\#load + \#store}{\#pixels\_per\_frame} \tag{1}$$

Computation: According to formula (2), the calculation instruction number of each pixel is used to represent the calculation amount required to complete the current algorithm operation. Among them, #ins represents the total number of current instructions, #branch represents the number of branch instructions.

$$compute = \frac{\#ins - \#load - \#store - \#branch}{\#pixels\_per\_frame} \tag{2}$$

Power consumption: With the high performance and low power consumption requirements of graphics processors, power consumption is an important factor in graphics rendering. The power consumption of each pixel can be calculated according to formula (3).

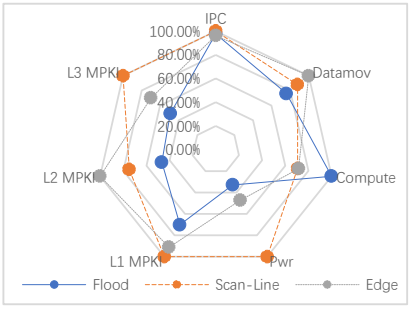
$$pwr = \frac{power\_all}{\#pixles\_per\_frame} \tag{3}$$

MPKI: MPKI is a general indicator that represents the average number of misses per thousand instructions. This indicator combines the advantages of cache hit ratio and load, store or cache access. Although cache hits are much faster than hits in DRAM, they still result in performance loss.

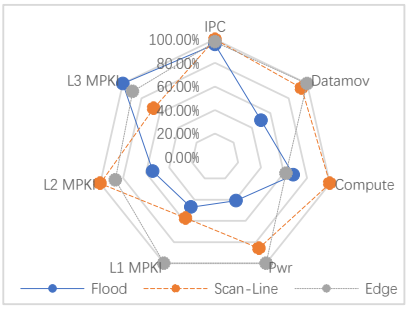
IV. Performance Characterization and Reconfiguration Suggestions

A. Overview of the Performance Characteristics

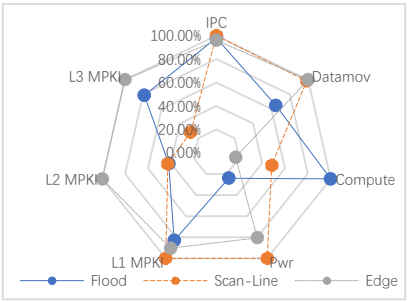
In order to systematically analyze the performance characteristics of different algorithms, this paper displays different indicators in the form of radar charts, as shown in Figure 2. The radar chart shows the performance indicators of the three rasterization algorithms of scan-line filling algorithm, edge filling algorithm and flood filling algorithm in four different rendering scenarios. The performance indicators in each figure include the algorithm's IPC, data movement, computation, power consumption, and L1, L2, and L3 data cache MPKI. Among them, the maximum value of each indicator is regarded as 1, and other data have been standardized.



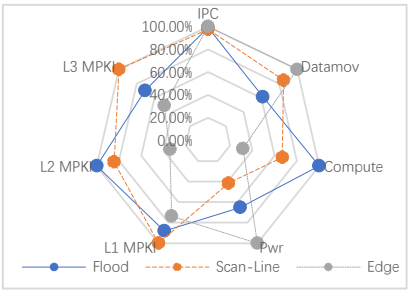
(a) Venus



(b) Cube



(c) Cartoon



(d) Teapot

Fig.2 Performance comparison of three algorithms in different scenarios

It can be seen from the figure that in different rendering scenarios, there is not much difference in IPC between different algorithms, which indicates that the potential parallelism of the three algorithms is similar. The edge filling algorithm shows higher data movement in most cases, while the flood filling algorithm has lower data movement in most cases, indicating that when the data movement becomes the system bottleneck, the flood filling algorithm is the best select. The edge filling algorithm has lower computational complexity in most cases, which shows that the edge filling algorithm is the best choice when hardware computing resources are limited. For power consumption, it can be observed that the power consumption generated by the flood filling algorithm is almost the least. Therefore, when power consumption is very important, in order to reduce the power consumption of the graphics processor, you can choose the flood filling algorithm. At the same time, it can be found that the flood filling algorithm has a smaller cache MPKI than the other two algorithms. Therefore, the flood filling algorithm is the best choice when the cache hit rate needs to be improved.

The above performance data is only an overview of the performance characteristics of different algorithms. It can only show the relative data of each performance index, and cannot accurately show the performance index of each algorithm. The detailed measurement data will be analyzed below.

B. Data Movement

Table II shows the data movement of each algorithm in the four different rendering scenarios. It can be seen that tens of thousands of instructions are needed to complete the data movement to render each pixel. In most cases, the flood filling algorithm has less data movement, and the edge filling algorithm has the relatively most data movement. At the same time, it can be found that the scan-line filling algorithm has similar data movement to the edge filling algorithm. Therefore, in order to improve GPU performance, the scan-line filling algorithm or the edge filling algorithm can be switched to the flood filling algorithm, so that the rasterization module has the least amount of data movement.

Table II Data movement per pixel of the three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	18,678	21,616	24,554
Cube	54,658	102,692	109,317
Cartoon	10,886	16,521	16,777
Teapot	42,108	57,961	68,364

C. Computation

The computation of each algorithm in the four different rendering scenarios is shown in Table III. Similar to data movement, thousands of operations are required to complete the rendering of one pixel. It can be seen that the flood filling algorithm of the three algorithms has a larger amount of computation, while the edge filling algorithm has fewer calculation operations. Therefore, when the rendering scene is limited by hardware computing resources, you can switch the scan-line filling algorithm or the flood filling algorithm to the edge filling algorithm, which can save approximately 45.53% of the computation.

Table III Computation per pixel of three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	17,343	12,256	12,390
Cube	23,883	35,014	21,664
Cartoon	12,331	5,968	2,075
Teapot	37,966	25,383	11,968

D. Power Consumption

In order to compare the power consumption of these three algorithms, this paper counts the total power of each algorithm running in different rendering scenarios, which shows the energy consumed each time the program is executed (unit: mJ). As shown in Table IV, although the calculation amount of the flood filling algorithm is relatively large, it consumes almost the least energy. Therefore, when power consumption is very important, in order to reduce the power consumption of the graphics processor, a flood filling algorithm can be selected. In addition, when the power consumption of the edge filling algorithm is large, switching between the flood filling algorithm and the scan-line filling algorithm is also a solution to reduce power consumption.

Table IV Power consumption per pixel of three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	0.0050	0.0153	0.0072
Cube	0.0042	0.0088	0.0103
Cartoon	0.0009	0.0038	0.0030
Teapot	0.0180	0.0115	0.0276

E. MPKI

The L1, L2 and L3 data cache MPKI of the three algorithms are shown in Table V, Table VI, and Table VII, respectively. It can be seen that in all cases, the L1 data cache MPKI does not exceed 10, the L2 data cache MPKI is almost half of the L1 data cache MPKI, and the L3 data cache MPKI is less and can be almost ignored. Therefore, in graphics rendering hardware, only L1 data cache is sufficient. It can be found in the L1 data cache MPKI that, although the flood filling algorithm has a larger amount of computation, it has a smaller cache MPKI compared to the other two algorithms. Therefore, choosing the flood filling algorithm can improve the cache hit rate, and it can be concluded that the flood filling algorithm uses a smaller MPKI to process more computing operations.

Table V L1 data cache MPKI of three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	2.61	3.73	3.40
Cube	4.35	5.30	9.22
Cartoon	6.61	7.96	7.19
Teapot	5.05	5.73	4.22

Table VI L2 data cache MPKI of three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	0.92	1.47	1.96
Cube	1.63	3.01	2.62
Cartoon	1.30	1.33	3.12
Teapot	2.40	2.03	0.82

Table VII L3 data cache MPKI of three algorithms

Dataset	Flood	Scan-Line	Edge
Venus	0.18	0.37	0.26
Cube	0.72	0.48	0.65
Cartoon	0.94	0.34	1.20
Teapot	0.34	0.48	0.23

F. Reconfiguration Suggestions

Based on the collected experimental data of a large number of different performance indicators, the Pearson correlation coefficient (PCC) [23] method is used to conduct a more in-depth correlation performance analysis. Table VIII lists the parameters that each index has on performance and energy consumption. The parameter range is between +1 and -1. +1 means positive correlation, -1 means negative correlation, and 0 means no correlation. As can be seen from the table, data movement and cache instruction missing have the highest correlation with performance and energy consumption, and should be the focus of algorithm optimization.

Table VIII Correlation of performance and energy to different metrics

Metrics	Correlation to Performance	Correlation to Energy
IPC	1.000	-0.267
Data movement	-0.553	0.368
Computation	-0.537	0.259
Energy Consumption	-0.267	1.000
L1 MPKI	-0.073	-0.270
L2 MPKI	-0.316	-0.176
L3 MPKI	-0.151	-0.521

Based on the analysis of the above data, this paper proposes some reconstruction suggestions for the rasterization module of the reconfigurable graphics processor:

a. When the amount of data movement becomes the system bottleneck, the scan-line filling algorithm or edge filling algorithm can be switched to the flood

filling algorithm to improve the performance of the GPU. The switching basis can be obtained by designing a counter to count the data movement of accessing the memory.

b. When the rendering scene is limited by hardware computing resources, the scan-line filling algorithm or the flood filling algorithm can be switched to the edge filling algorithm, and the switching basis can be obtained by setting a counter in the execution module.

c. When power consumption is very important, in order to reduce the power consumption of the graphics processor, you can choose the flood filling algorithm. In addition, when the power consumption of the edge filling algorithm is large, switching between the flood filling algorithm and the scan-line filling algorithm is also a solution to reduce power consumption.

d. In the past few decades, the cache technology has made great progress. In many cases, the cache has less MPKI. Therefore, in the architecture of reconfigurable graphics hardware, a first-level cache is sufficient. At the same time, choosing the flood filling algorithm can improve the cache hit rate.

Based on the above conclusions, three counters can be inserted into the rasterization module, and the load instructions, store instructions and branch instructions executed by them can be counted respectively. Whenever the respective instruction enable signal is monitored as 1, the corresponding counter is increased by 1. The statistical results are calculated according to formula (1) and formula (2) through the adder and the subtractor to obtain the data movement and calculation amount of the rasterization module, and the most suitable algorithm in the current state is selected according to the reconfiguration suggestions. The H-tree hierarchical configuration network HRM [23] can be used to configure and deliver different rasterization algorithms, and use the reconfigurable array processor [24] to realize the reconfigurable design of the rasterization module in the GPU.

V. CONCLUSIONS

Rasterization is an important part of the graphics processor, always requires a lot of operations, and is a performance bottleneck. Rasterization algorithms usually suffer computational constraints, memory constraints, and power consumption constraints in different application scenarios, so it is important to determine how to independently schedule different rasterization algorithms based on actual needs. However, the reconfiguration standard for graphics processors has not been established. Therefore, this paper evaluates and analyzes the performance characteristics of three rasterization algorithms (scan-line filling algorithm, edge filling algorithm, and flood filling algorithm) in different application scenarios. Pearson correlation coefficient (PCC) analyzes the relationship between performance/energy and evaluation metrics. Based on these performance characterization data, this paper puts forward some reconstruction suggestions in the reconfigurable graphics processor: When the amount of data movement becomes a system bottleneck or power consumption is very important, the flood filling algorithm can be selected to improve the performance of the GPU. When hardware computing resources are limited, the edge filling algorithm is the best choice. In addition, choosing the flood filling algorithm can improve the cache hit rate. And you can insert a counter in the rasterization module, set up a state monitoring module, etc. to realize the reconfigurable design of the rasterization module.

ACKNOWLEDGMENT

This work is supported in part by National Science Foundation of China under Grant 61834005, 61772417, 61802304, and 61874087, International S&T Cooperation Program of Shaanxi China under Grant 2018KW-006.

REFERENCES

[1] Xie C, Fu X, Song S, “Perception-oriented 3D Rendering Approximation for Modern Graphics Processors,” 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018, pp. 362-374.

[2] Christian K, Tobias S, Wolfgang B. “Efficient Point Cloud Rasterization for Real Time Volumetric Integration in Mixed Reality Applications,” 2018 IEEE International Symposium on Mixed and Augmented Reality. IEEE, 2018, pp. 1-9.

[3] Nguyen-Phuoc T, Li C, Balaban S, et al. “RenderNet: A Deep Convolutional Network for Differentiable Rendering from 3D Shapes,” 2018.

[4] Tao H, Manchun L, Xiaohui J, et al. “Implementation of Parallel Algorithm for Rasterization Processing,” 2012 IEEE,20th International Conference on Geoinformatics. pp. 1-5.

[5] Akenine-Möller T, Haines E, “Hoffman N. Real-time Rendering (Third Edition),” CRC Press, 2008, pp. 697-725.

[6] Deng J, Jiang L, Cui J. “An Energy Efficiency-driven Programmable and Self-reconfigurable Architecture of Mobile Graphics Processor,” International Conference on Computer Science and Information Security. 2016, pp. 3639-3657.

[7] Issa J, Figueira S. “Graphics Processor Performance Analysis for 3D Applications,” Proceedings of the 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA’12), Dec 12-15, 2012, Beirut, Lebanon. Piscataway, NJ, USA: pp. 269 -272.

[8] Patel M, Hubbold R J. “A Scanline Method for Solid Model Display,” John Wiley & Sons, Ltd. 1987, pp. 141-150.

[9] Lejun S, Hao Z. “A New Contour Fill Algorithm for Outlined Character Image Generation,” Computers & Graphics, vol. 19, no. 4, pp. 0-556, 1995.

[10] Arvo J, Hirvikorpi M. “Approximate Soft Shadows Win an Image-space Flood-fill Algorithm,” Computer Graphics Forum, vol. 23, no. 3, pp. 271-279, 2004.

[11] Wei S, Liu L, Yin S. “Key Techniques of Reconfigurable Computing Processor,” Scientia Sinica (Informationis), vol. 42, no. 12, pp. 79-96, 2012.

[12] Chen Y, Krishna T, Emer J, et al. “Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, 2017.

[13] Liu L, Zhou Z, Wei S, et al. “DRMaSV: Enhanced Capability against Hardware Trojans in Coarse Grained Reconfigurable Architectures,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017, 37(4):782-795.

[14] Kotra J B, Zhang H, Alameldeen A R, et al. “Chameleon: A Dynamically Reconfigurable Heterogeneous Memory System,” 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018, pp. 533-545.

[15] Prabhakar R, Zhang Y, Koeplinger D, et al. “Plasticine: A Reconfigurable Accelerator for Parallel Patterns,” IEEE Micro, vol. 38, no. 3, pp. 20-31, 2018.

[16] Gao M, Yang X, Pu J, et al. “Tangram: Optimized Coarse-grained Dataflow for Scalable an Accelerators,” Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2019, pp. 807-820.

[17] Liu L, Zhu J, Li Z, et al. “A Survey of Coarse-grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications,” ACM Computing Surveys, vol. 52, no. 6, pp. 1-39, 2019.

[18] Cutress, Ian. “The AnandTech Coffee Lake Review: Initial Numbers on the Core i7-8700K and Core i5-8400,” https://www.anandtech.com/show/11859/the-anandtech-coffee-lake-review-8700k-and-8400-initial-numbers.

[19] Intel R. “Intel® 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4,” https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.

[20] Jiang L, Chen L, Qiu J. “Performance Characterization of Multi-threaded Graph Processing Applications on Many-integrated-core Architecture,” 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2018, pp. 199-208.

[21] Reinders J. “VTune Performance Analyzer Essentials: Measurement and Tuning Techniques for Software Developers,” Intel Press, vol. 1, no. 2, pp. 6, 2005.

[22] Deng J, Liu Y, Xie X. “Performance characterization of illumination algorithms for reconfigurable graphics processor,” Journal of China Universities of Posts & Telecommunications, vol. 26, no. 5, pp. 60-71, 2019.

[23] Benesty J, Chen J, Huang Y, et al. “Pearson Correlation Coefficient,” Noise reduction in speech processing. Springer, Berlin, Heidelberg, 2009, pp. 1-4.

[24] Jiang L, Deng J, Song S, et al. “Hrm: H-tree Based Reconfiguration Mechanism in Homogeneous PE Array for Video Processing,” 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). p. poster, IEEE, 2018.

[25] Zhu Y, Jiang L, Shi P F, et al. “Parallelization of Intra Prediction Algorithm Based on Array Processor,” High Technology Letters, vol. 25, no. 1, pp. 74-80, 2019.