HIGH TECHNOLOGY LETTERS | Vol. 30 No. 1 | Mar. 2024 | pp. 31-42

doi:10.3772/j.issn.1006-6748.2024.01.004

BAR: a branch-alternation-resorting algorithm for locality exploration in graph processing①

DENG Junyong(邓军勇)*, WANG Junjie②*, JIANG Lin**, XIE Xiaoyan***, ZHOU Kai*

(*School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, P. R. China)

(**School of Computer, Xi'an University of Science and Technology, Xi'an 710054, P. R. China)

(***School of Computer, Xi'an University of Posts and Telecommunications, Xi'an 710121, P. R. China)

Abstract

Unstructured and irregular graph data causes strong randomness and poor locality of data accesses in graph processing. This paper optimizes the depth-branch-resorting algorithm (DBR), and proposes a branch-alternation-resorting algorithm (BAR). In order to make the algorithm run in parallel and improve the efficiency of algorithm operation, the BAR algorithm is mapped onto the reconfigurable array processor (APR-16) to achieve vertex reordering, effectively improving the locality of graph data. This paper validates the BAR algorithm on the GraphBIG framework, by utilizing the reordered dataset with BAR on breadth-first search (BFS), single source shortest paht (SSSP) and betweenness centrality (BC) algorithms for traversal. The results show that compared with DBR and Corder algorithms, BAR can reduce execution time by up to 33.00%, and 51.00% seperatively. In terms of data movement, the BAR algorithm has a maximum reduction of 39.00% compared with the DBR algorithm and 29.66% compared with Corder algorithm. In terms of computational complexity, the BAR algorithm has a maximum reduction of 32.56% compared with DBR algorithm and 53.05% compared with Corder algorithm.

Key words: graph processing, vertex reordering, branch-alternation-resorting algorithm (BAR), reconfigurable array processor

0 Introduction

Graph processing is widely used in many important fields, from social networks to bioinformatics[1], from power grid management to criminal network detection[2], etc. Graph processing is developing rapidly and deeply. The industry and academia have carried out some promising research and exploration. However, the large-scale unstructured and irregular graph data causes strong randomness and poor locality which limit the efficiency of graph processing systems. Multiple types of input graph data make a graph processing system suffer from input dependencies and show a huge performance gap.

Conventional wisdom holds that graph algorithms are inherently random access. Specifically, they have a strong community structure, corresponding to communities in the real world. In addition, vertices are more popular than other vertices in many graphs, so they are accessed more frequently, and graph algorithms therefore provide potential locality[3]. This locality can often be explored and analyzed through traversal operations, trarevsal operations are the order in which the vertices and edges of the graph are processed. However, the existing locality model runs at the program level of regular loops and arrays, or runs at the trace level of arbitrary access streams, so they are not sufficient to characterize the relationship between locality and connectivity. The efficiency of graph computing depends heavily on the bandwidth between processors, because the graph structure is sent over the network after each iteration. Although much data may remain the same between iterations, the data must be reloaded and reprocessed with each iteration, fetching a high computational ratio. This results in unnecessary I/O, poor data locality, wasted network bandwidth and processor resources.

① Supported by the National Key R&D Program of China (No. 2022ZD0119001), the National Natural Science Foundation of China (No. 61834005), the Shaanxi Province Key R&D Plan (No. 2022GY-027), the Key Scientific Research Project of Shaanxi Department of Education (No. 22JY060), the Education Research Project of XUPT (No. JGA202108) and the Graduate Student Innovation Fund of Xi'an University of Posts and Telecommunications (No. CXJJZL2022011).

② To whom correspondence should be addressed. Email: wangjunjie9706@163.com.

Received on May. 6, 2023

In order to alleviate the problem of low graph processing efficiency, previous work has designed different graph processing accelerators[4] to improve graph processing efficiency. However, graph processing accelerators often only effectively accelerate some graph calculations, and often cannot achieve the acceleration effect when faced with other graph data or graph calculations. Therefore, Researchers have focused on the preprocessing of graph data for acceleration. By mining the locality of graph data in graph processing, a good acceleration effect is achieved. In recent years, a large number of algorithmic ideas have emerged around the preprocessing of graph data, including popular graph partitioning algorithms[5] and graph vertex reordering algorithms[6]. This paper will preprocess graph data in graph processing based on the idea of reordering graph vertices to alleviate the problem of poor locality of graph data. Because breadth-first search (BFS)[7] and depth-first search (DFS)[8] are the two most widely used traversal methods in graph data sorting, they can effectively optimize the locality of most graph data. Due to the fact that some vertices in graph data are more popular and accessed more frequently than others, in the previous work, the traversal advantages of BFS and DFS are to propose a depth-branch-resorting algorithm (DBR)[9], which effectively alleviates the problem of poor locality of graph data in graph processing. However, after a large number of analysis and experiments, vertices in graph data have a strong community structure. How to maximize the mining of graph data locality will be the focus of this paper.

In order to alleviate the problem of poor locality of graph data in the process of graph processing , this paper optimizes the DBR, and proposes a branch-alternation-resorting algorithm (BAR). The BAR algorithm is mapped onto the reconfigurable array processor (APR-16)[10], effectively alleviating the challenges of poor graph data locality and high memory access processing rate. Finally, the results are applied to the GraphBIG[11] framework to verify and analyze the results.

In this paper, Section 1 introduces and analyzes the development status of graph computing in recent years. In Section 2, the BAR algorithm is proposed and its traversal method is introduced. Section 3 introduces the mapping scheme of the BAR algorithm on APR-16. In Section 4, the performance of the BAR algorithm and the mapped scheme is evaluated. In Section 5, the paper is summarized.

1 Background and motivation

In recent years, many important issues in social network analysis, artificial intelligence, business analysis[12], and computational science require graph processing. However, optimizing the locality problem of graph processing is a challenging task[13] due to the size of large graphs and the inherently irregular structure of graphs. Based on this, many researchers have carried out in-depth research on it. Combining the storage characteristics, memory accesses characteristics, and data locality of different processing platforms, researchers improve the locality by designing corresponding data organization formats, graph vertex reordering scheme and graph partitions.

The graph reordering algorithm optimizes the data layout and calculation order by changing the index order of vertices without changing their underlying connections, and improves the locality of graph data access and processing. The graph reordering algorithm can accelerate a large number of mainstream graph algorithms without modifying each algorithm itself. Complex graph reordering algorithms greatly speed up graph processing while incurring significant computational overhead.

Ref. [14] introduced a new metric that quantifies cache data reuse, resulting in a heuristic pH value that enhances temporal locality in the memory access patterns (MAP) of the graph algorithm. A concept of dynamically matching MAP and cache content is defined, which can jointly maximize cache data reuse and cache line utilization. Ref. [15] rearranged the vertices in the graph in decreasing order (referred to as degree sorting), which was thoroughly motivated and described as RADAR. This is a system that combines duplication and reordering into a single graph processing optimization, reaping the benefits and eliminating the costs of both. RADAR improves performance of graph applications by reducing the number of atomic updates and improving locality of memory accesses.

The bounded depth first scheduling (BDFS) proposed by Ref. [3] is an online locally aware scheduling strategy. A hardware accelerated traversal scheduler (HATS) is designed to improve graph locality. Ref. [16] proposed a cache simulation technique for processing large graphs and a spatial location metric called neighbor to neighbor average ID distance to investigate how graph reordering algorithms affect the positions of different vertices. Ref. [6] proposed a community aware reordering (CAR) algrithm, which finds the community structure using breadth-first traversal and takes advantage of the power-law nature exhibited by real-world graphs by coarsely binning them into different groups to preserve the explored graph structure.

Ref. [17] proposed a cache aware graph reorde-

ring algorithm called Corder, which implements parallelism by using a dynamic scheduler to unroll loops. In other words, loop iterations are dynamically allocated to each thread to achieve workload balancing. Based on a dynamic scheduling strategy, the Corder algorithm further fine-tunes the computational cost of each iteration through graph reordering.

In order to alleviate the problem of poor locality of graph data during graph processing, based on the idea of graph vertex reordering algorithm, this paper optimizes the DBR[9], and proposes BAR algorithm. The BAR algorithm is mapped onto the reconfigurable array (APR-16)[18] to achieve parallel acceleration of algorithm operation.

2 The BAR algorithm

2.1 The DBR algorithm

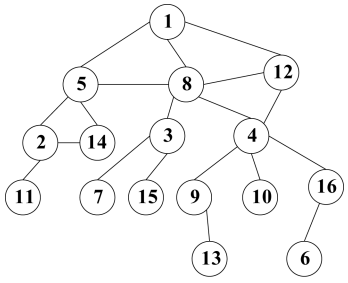
Distributed graph processing systems increase locality by studying data organization formats, resorting graph data, or dividing graphs. Sophisticated graph resorting techniques can effectively reduce the runtime of graph applications, but the resorting step also brings corresponding computational overhead. DBR combines the advantages of hierarchical community mining and deep community mining, and can reduce the computational overhead of data preprocessing and algorithm operation.

In hierarchical locality community mining, locality mining is performed hierarchically until the entire graph is fully traversed. In depth locality community mining, the locality of graph data is mined branch by branch of a specified depth, and then other branches are explored until the entire graph is completely traversed.

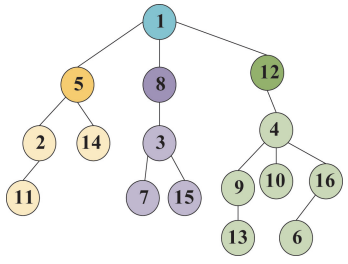
To achieve the above-mentioned purpose, the DBR algorithm comprises two parts: the first part is conducted through hierarchical local community mining, with the ability to automatically adjust the number of layers for partitioning; the second part involves a depth-first traversal based on the branches of the nodes identified in the first part. The fundamental idea is as follows: conduct hierarchical traversal on the source node s and keep track of the traversed layers. When the specified number of layers is reached, perform a depth-first traversal on the remaining graph data. The process diagram of the DBR algorithm is illustrated in Fig. 1.

2.2 The BAR algorithm

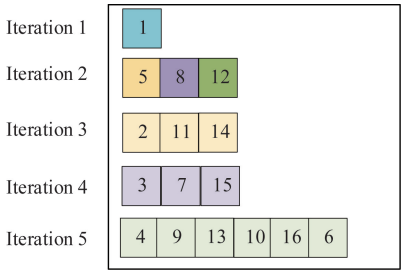
In order to reduce the amount of computation and execution time of graph data processing during graph processing, this paper optimizes the DBR algorithm, aiming to further explore the community locality of graph data. In this paper, combining the advantages of breadth first search and depth first search, and based on the characteristics of different graph data, corresponding traversal levels will be specified to alternately perform breadth first search and depth first search to realize the improvement of local differences in graphics data.



(a) Sample graph



(b) DBR tree

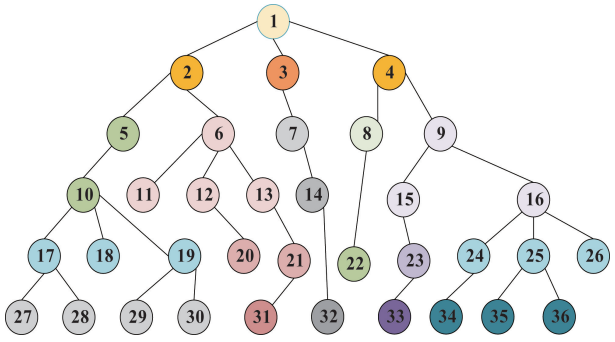


(c) DBR ordering of a layer of child nodes

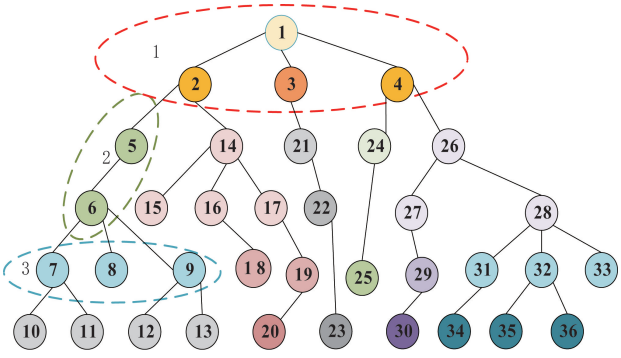
Fig. 1 DBR algorithm vertex reordering process

In order to achieve the above objectives, the idea adopted in this paper is to first use the breadth first search traversal method for the source node *s*. After reaching the specified number of layers, the first node *j* of the last layer previously traversed is traversed using the depth first search traversal method. After reaching the specified number of layers, the last node is used as the source node to continue the breadth first search. After reaching the specified number of layers, perform a depth first search on the vertices of that number of layers at once. After all traversal of node *j* is completed, start traversing other nodes in the same manner.

As shown in Fig. 2, an example of 36 vertices is shown. This paper sets the number of breadth first search layers to 1, and the number of depth first search layers is set to 2. Start traversing from source node 1, and first perform a breadth-first search. Traverse nodes 2, 3, and 4 in turn. Then, starting with node 2 as the source node, a depth first search is performed through two layers, traversing nodes 5 and 6. Conduct a layer of breadth first search with node 6 as the source node. Perform a depth first search on nodes 17, 18, and 19 in turn. After that, the other branches of node 2 are traversed in the same manner. After the traversal of node 2 is completed, the other child nodes of node 1 are traversed in the same manner. The example graph traversal vertex order in Fig. 2(a) is shown in Fig. 2(b).



(a) 36-node example diagram



(b) Example graph vertex traversal order

Fig. 2 BAR algorithm vertex reordering process

3 Mapping of BAR on APR-16

BAR algorithm is a serial algorithm, which often takes a long time to be executed and has low efficiency when processing large graph data. The problem of poor locality of graph data during graph processing is difficult to effectively solve. To solve the problem of low runtime efficiency of software algorithms, this paper conducts parallel analysis of the algorithm and maps it onto the reconfigurable array (APR-16)[19]. By activating multiple processing element(PE) pairs simultaneously, the algorithm can run in parallel to improve the efficiency of the algorithm.

3.1 The reconfigurable array

The processing core of the reconfigurable video array processor is a PE array composed of 16 PEs. The PE array is mainly used to perform processing functions. At the same time, each PE corresponds to a data bank, and 16 distributed data banks are used to cache the intermediate results generated by the calculation. The global unified addressing method is adopted, and the row and column two-level switching unit is used to achieve PE access to any data bank.

A single cluster reconfigurable array processor consists of one 4 × 4 PE array, 16 distributed data banks, H-tree[19] instruction transmission network, input first in first out(FIFO), output FIFO, instruction/data distribution controller, data recovery controller, status register, input data buffer, output data buffer, frame buffer, and array data judgment unit. The overall structure is shown in the Fig. 3.

The 4 ×4 PE array is mainly used to perform processing functions. The 16 distributed data banks are used to cache intermediate results generated by calculations. The input data FIFO is used to cache data information from the upper computer, and the output data FIFO is used to send the results of the internal calculation of the chip to the upper computer. The input data buffer is used to store the code blocks to be processed, the output data buffer is used to store the data blocks after the calculation is completed, and the frame buffer is used to store the reference frame pixels required for the next frame after the calculation is completed. The status register is composed of three control bits, namely I, D, and C. I is used to indicate that the currently transmitted instruction information; D is used to indicate that the currently transmitted is coded block data information; C is used to configure the test circuit and read out the initialized configuration information from the test circuit.

3.2 Algorithm mapping scheme

Due to the large amount of graph data and strong community in the graph calculation process, the main issues affecting the energy efficiency of graph calculation are the large amount of graph data. Therefore, in the process of algorithm mapping, this paper divides the mapping process into two parts: graph data preprocessing and reordering, aiming at the performance impact of the large amount of graph data on the algorithm.

3.2.1 Preprocessing of graph data

To mitigate the impact of a large graph dataset on

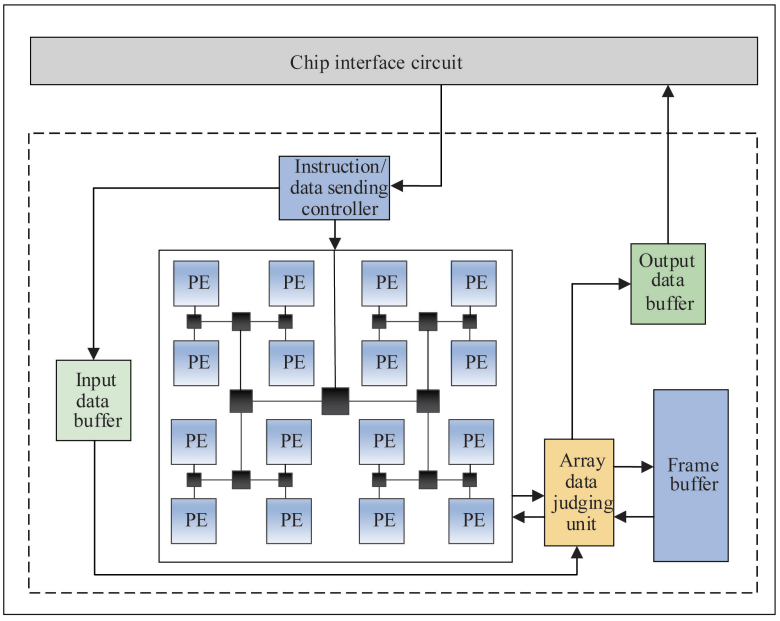


Fig. 3 Graph data preprocessing PE mapping diagram

the reordering process, this paper allocates 14 PEs to preprocess all vertices in the graph dataset, as shown in Fig. 4. Find the adjacent points of all nodes, and store the found node information in FRAME.

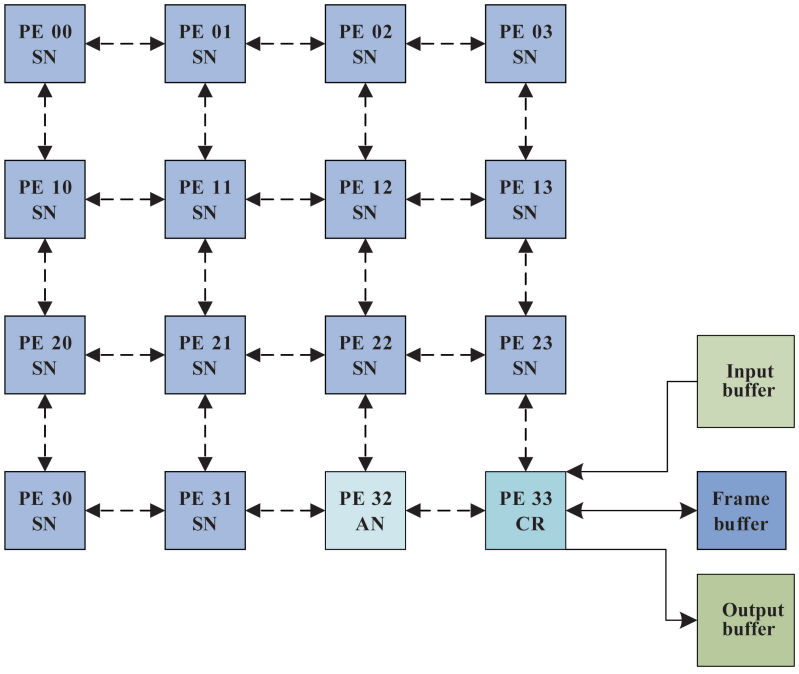


Fig. 4 Graph data preprocessing PE mapping diagram

In APR-16, PE33 is responsible for cyclically reading graph data (CR), PE32 is responsible for assign nodes (AN), and the remaining PEs are responsible for search nodes (SN). PE33 connects input buffer, frame, and output buffer. Iterate through PE33 to retrieve data. PE32 finds all adjacent points of the source node, and stores the node in FRAME. After that, PE32 is responsible for sending unprocessed nodes to the remaining 14 PEs, and each PE finds all adjacent points of each allocation node by looking for the data input in the loop in PE33. After all PEs have processed a set of vertex data, the data is stored in FRAME, and the PE32 resends the new unprocessed node data, repeating the work. When the graph dataset has a large amount of data, the data in FRAME is output to external storage through PE33. The preprocessing of graph data is comple-

ted until all neighbor nodes are found for each vertex in the graph data. The flowchart is shown in Fig. 5.

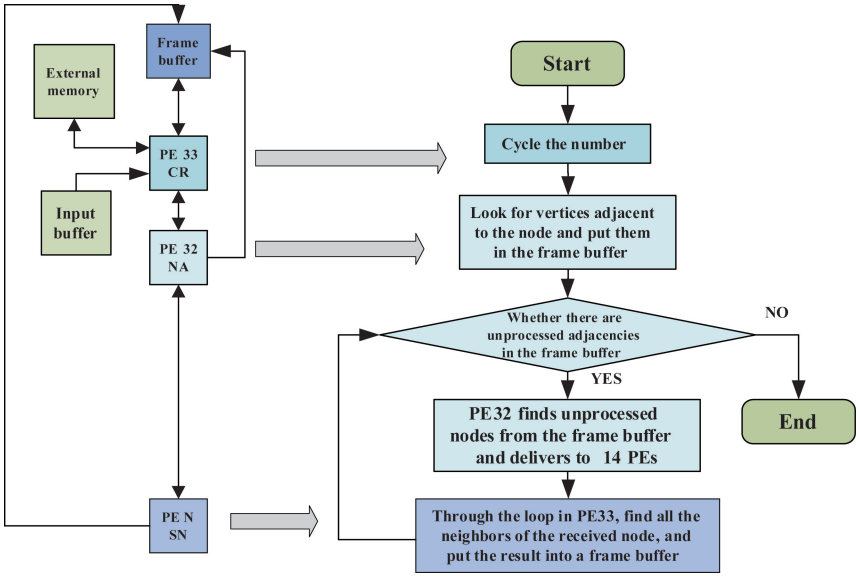


Fig. 5 Graph data preprocessing mapping flowchart

3.2.2 Vertex reordering

After the data and processing are completed, the adjacent point information of each vertex in the graph dataset is in FRAME, and begin the reordering process. The functional structure design of PE is shown in Fig. 6.

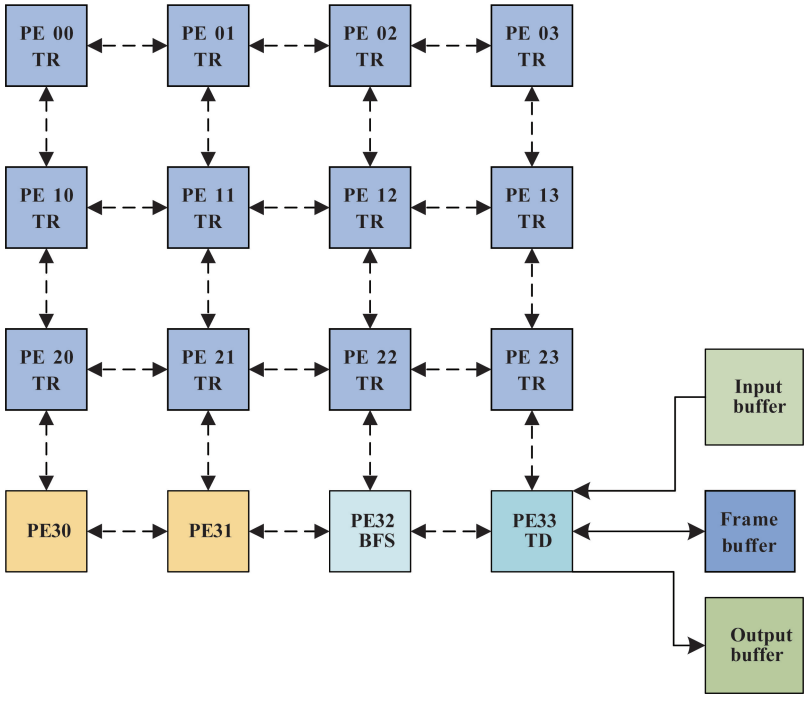


Fig. 6 Vertex reordering PE mapping diagram

In Fig. 6, PE33 is responsible for transmitting data (TD), PE32 for BFS vertex traversal based on the source node, PE30 and PE31 for data distribution, and the remaining PEs are responsible for vertex traversal (TR). PE33 transmits the data required by each PE from the FRAME and external memory. After the algorithm starts executing, PE30 first starts traversing the original vertices in BFS traversal mode. When the specified number of layers is reached, the traversal results are stored in FRAME, and all vertices within the cur-

rent number of layers are placed in the data bank. PE30 and PE31 distribute the vertex data in the bank to 12 PEs, and begin alternating depth and breadth traversal. In traversing the PE, the reordering of the branches starts with the received node as the source node. After the sorting is completed, the processing result is sent to the frame buffer, and an end signal is sent to PE30 and PE31. Wait for PE30 and PE31 to send a new node and restart the traversal. After all nodes in PE30 and PE31 have been distributed and traversed, the algorithm will be executed, and the reordered vertices can be taken out in FRAME. The flowchart of vertex reordering is shown in Fig. 7.

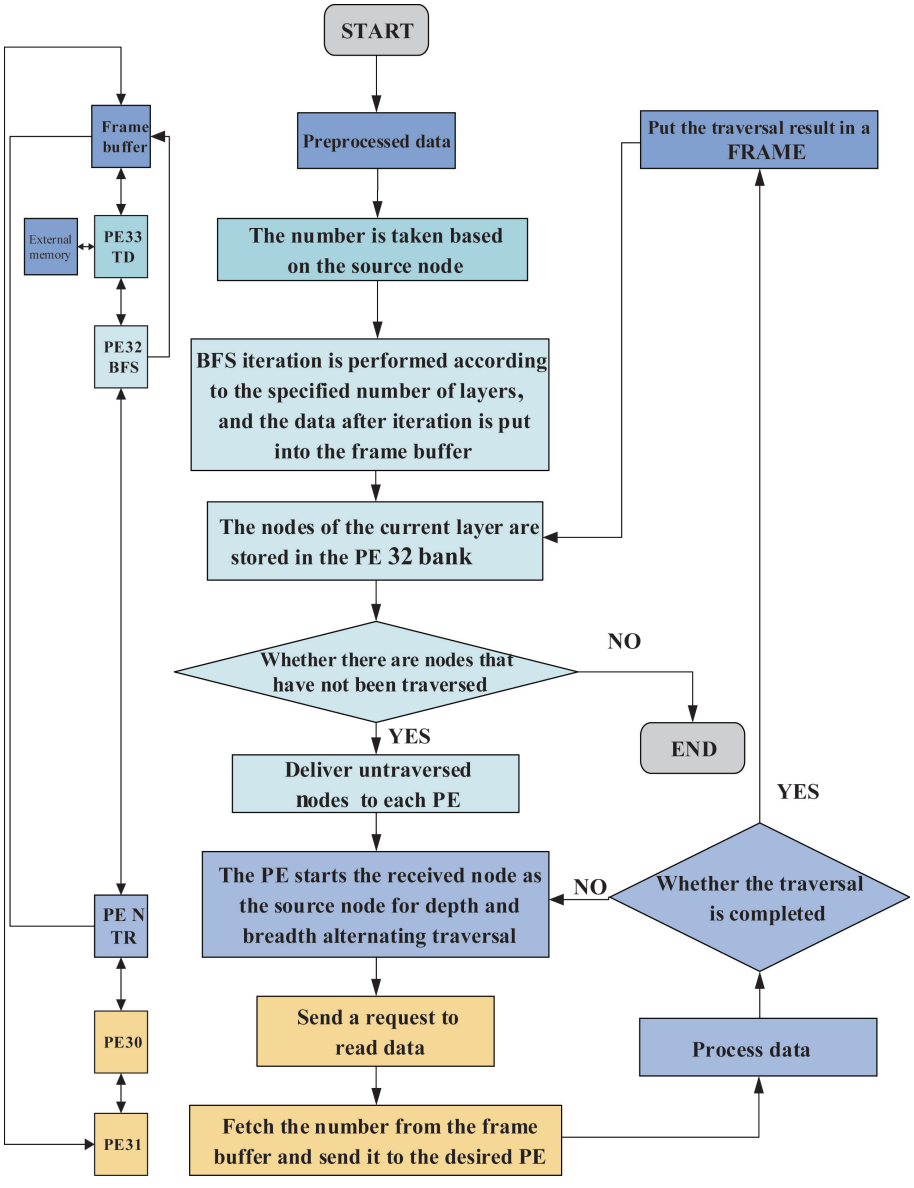


Fig. 7 Vertex reordering mapping flowchart

4 Experimental results and analysis

4.1 Experiment setup

This paper selects the graph processing framework GraphBIG[10] for testing and validation. GraphBIG is a graph benchmarking effort initiated by Georgia TechHP Arch, which is a comprehensive set of graph processing tools and solutions for big data. GraphBIG includes representative benchmarks for both CPU and GPU to achieve an overall view of general graph processing. Fig. 8 shows the overall design of a verification scheme based on the GraphBIG framework.

The experiments in this paper are performed on an HPE580 high-performance server equipped with an Inter(R) Xeon(R) Platinum 8164 CPU. With 208 physi-

cal cores and 416 threads, each core has a 32 kB L1-level data cache, a 32 kB L1-level instruction cache, a 1 024 kB L2-level cache, and a 36 608 kB L3-level cache, with a memory size of 1 TB, running Linux kernel 4.15.0 system. In order to test more accurately, the ubuntu16.04 version is installed on the server, and the compiler is gcc 5.5.0 version. This chapter uses the evaluation tool Perf to perform parameter statistics on different real graph data onto performance events of traversal applications based on GraphBIG.

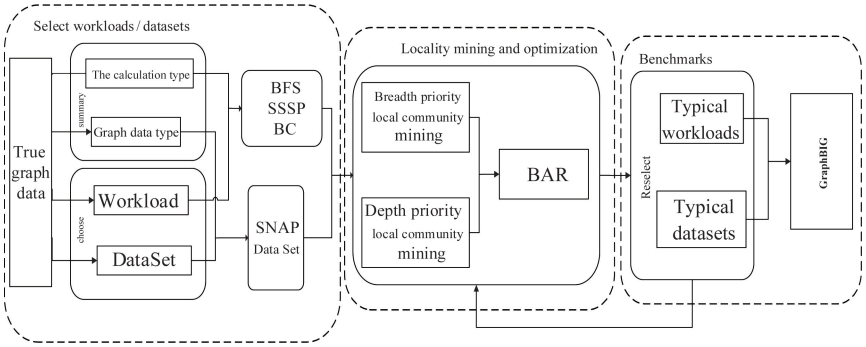


Fig. 8 GraphBIG framework verification scheme design

4.2 Experimental results

The graph data set used in the paper is selected from the Stanford Network Analysis Platform (SNAP). SNAP[20] dataset includes feather-deezer-social (Deezer), Wiki-Vote (Wiki) of social networks, ca-Astro-Ph (CA) in Collaboration networks, Soc-brightkite (Bright), Soc_gemsec_HU (HU) and Soc_gemsec_HR (HR) of social networks in the network data repository with interactive graph analytics and visualization dataset[21]. Table 1 lists the information for the node scale and edge scale of the selected graph dataset. Five algorithms of BFS, DFS, DBR[8], Corder[18] and BAR are used to preprocess six different graph datasets, and the preprocessing results are applied to the three traversal algorithms based on BFS, single source shortest path (SSSP) and betweenness centrality (BC), and Perf tools are used for parameter statistics.

Table 1 Dataset information

	Nodes	Edges
Deezer	28 281	92 752
Wiki	7 115	103 689
CA	18 772	198 110
Bright	56 739	212 945
HU	47 538	222 887
HR	54 572	498 202

After experimental comparison, it is found that with the increase of algorithm complexity, the more breadth-first search layers and depth-first search layers, the better the acceleration effect. When the dataset size is large, the more depth-first search layers, the better the acceleration effect. In the BFS algorithm and SSSP algorithm, BAR sets the number of breadth first search layers to 1, and the number of depth first search layers to 2 for the best results. In the BC algorithm, BAR sets the number of breadth first search layers to 2, and the number of depth first search layers to 3 for the best results.

4.2.1 Execution time

The graph dataset selected in this paper is sorted by BFS, DFS, DBR, Corder, and BAR algorithms, and implemented in three traversal algorithms: BFS, SSSP, and BC. The execution time of each edge is counted. The three sorting results for traversal applications specifically show that after BAR sorting, each edge have the least execution time, followed by the DBR algorithm, and the effect of the Corder algorithm is worse than the first two. It has been proved that reordering and adjusting the data can effectively reduce the time between memory accesses and optimize locality, with the best optimization effect of BAR algorithm.

As shown in Fig. 9, in the application of BFS algorithm, the execution time of HR dataset sorted by BAR algorithm is 33.64% shorter than DBR algorithm, and the execution time of HR dataset sorted by Corder algorithm is 27.00% shorter than HR dataset sorted by BAR algorithm. The execution time of Wiki dataset sorted by BAR algorithm is 16.67% shorter than DBR algorithm, and 37.50% shorter than Corder algorithm. Compared with DBR algorithm, the execution time of HU data set after BAR algorithm is shortened by 8.42%, and the execution time of HU data set after Corder algorithm is increased by 1.16%.

As shown in Fig. 10, in SSSP algorithm applications, the execution time of Wiki dataset sorted by BAR algorithm is 30.65% shorter than DBR algo-

rithm, and 39.44% shorter than Corder algorithm. The execution time of the HU dataset sorted by the BAR algorithm is 2.94% shorter than DBR algorithm, and the execution time is 3.13% longer than Corder algorithm.

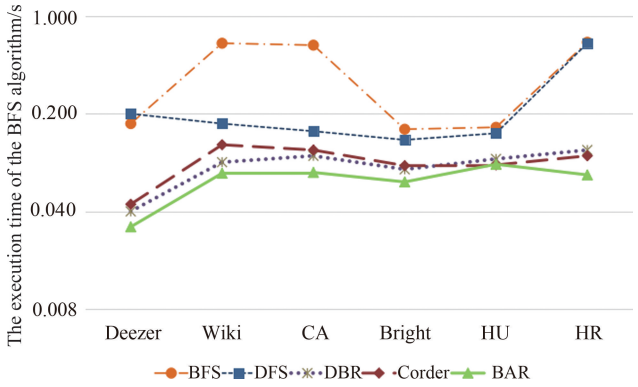


Fig. 9 The execution time of different vertex sorting methods on the BFS algorithm

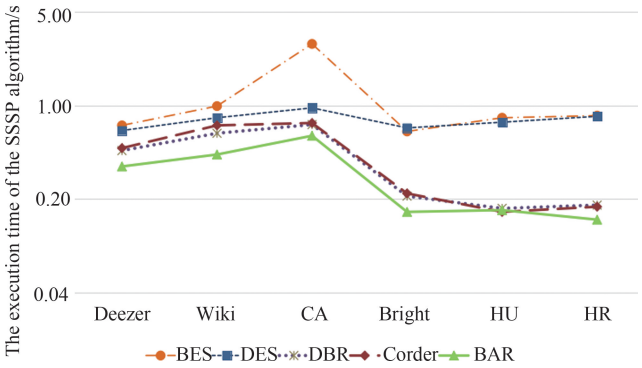


Fig. 10 The execution time of different vertex sorting methods on the SSSP algorithm

Due to the high complexity of the BC algorithm, the execution time of each edge is also long. As shown in Fig. 11, in the BC algorithm application, the execution time of Wiki dataset sorted by the BAR algorithm is 25.90% shorter than DBR algorithm, and the execution time is 33.51% shorter than Corder algorithm. The execution time of Bright dataset sorted by BAR algorithm is 24.39% shorter than DBR algorithm, and 51.56% shorter than Corder algorithm. The execution time of HU dataset sorted by BAR algorithm is 15.87% shorter than DBR algorithm, and 5.36% shorter than Corder algorithm.

From the above analysis, it can be seen that the BAR algorithm can effectively reduce the execution time of each edge. The maximum reduction is 37.50% compared with the DBR algorithm and 51.56% compared with the Corder algorithm.

4.2.2 Data movement

Six different graph datasets are sorted through BFS, DFS, DBR, Corder, and BAR algorithms, and then implemented in BFS, SSSP, and BC traversal algorithms. After that, data movement is counted. From the overall trend, the data movement amount of each side of the BC algorithm is higher than that of the BFS and SSSP algorithms. The three sorting results for traversal applications specifically show that after BAR sorting, the amount of data movement of the algorithm has been significantly reduced.

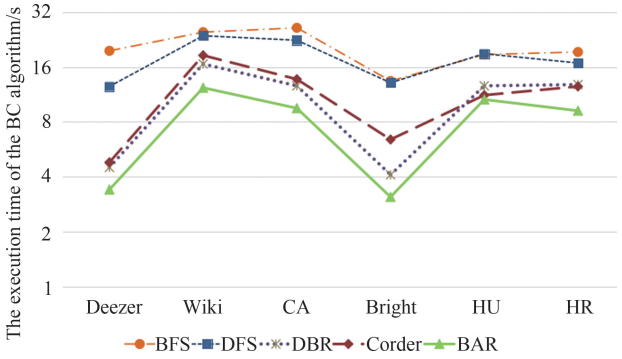


Fig. 11 The execution time of different vertex sorting methods on the BC algorithm

As shown in Fig. 12, in the application of BFS algorithm, data movement of Wiki data set after BAR algorithm sorting is reduced by 37.37% compared with DBR algorithm, and data movement after Corder algorithm sorting is reduced by 24.85%. Compared with DBR algorithm, the data movement volume of Bright dataset after BAR algorithm sorting is 21.18% lower than that after DBR algorithm sorting, and 26.77% lower than that after Corder algorithm sorting. The data movement of HR data set after BAR algorithm sorting is 22.53% lower than that after DBR algorithm sorting, and 4.18% higher than that after Corder algorithm sorting.

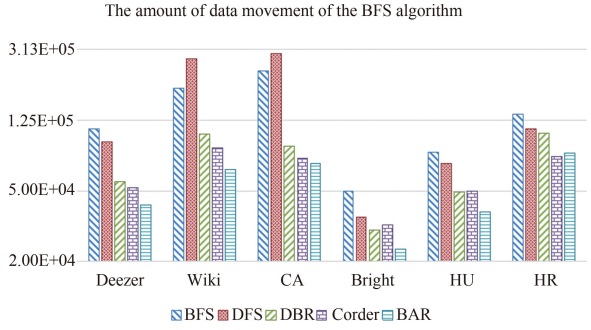


Fig. 12 The amount of data movement of different vertex sorting methods on the BFS algorithm

As shown in Fig. 13, in the application of SSSP algorithm, the data movement of HU dataset after sorting by BAR algorithm is reduced by 23.27% compared with DBR algorithm, and 10.26% lower than Corder algorithm. The data movement of HR dataset af-

ter sorting by BAR algorithm is reduced by 9.62% compared with DBR algorithm, and 1.91% lower than Corder algorithm.

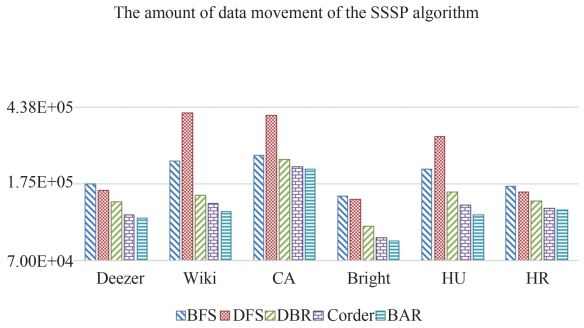


Fig. 13 The amount of data movement of different vertex sorting methods on the SSSP algorithm

As shown in Fig. 14, in the application of BC algorithm, the data movement volume of CA data set after BAR algorithm sorting is reduced by 39.00% compared with that after DBR algorithm sorting, and that after Corder algorithm sorting is reduced by 29.66%. The data movement of HR data set after BAR algorithm sorting is 4.95% lower than that after DBR algorithm sorting, and 1.08% higher than that after Corder algorithm sorting.

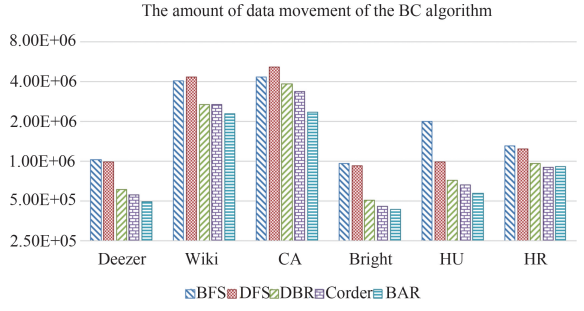


Fig. 14 The amount of data movement of different vertex sorting methods on the BC algorithm

From the above analysis, it can be seen that compared with DBR algorithm, BAR algorithm can effectively reduce the amount of data movement of the algorithm by up to 39.00%. Compared with Corder algorithm, BAR algorithm has a significant optimization effect in some data sets, with a maximum reduction of 29.66% in data movement.

4.2.3 Amount of computation

Six different graph datasets are sorted through BFS, DFS, DBR, Corder, and BAR algorithms, and then implemented in three traversal algorithms: BFS, SSSP, and BC. After that, the calculation amount of each edge is counted. The calculation amount is closely related to the number of instructions, the number of loaded and stored instructions. From the overall trend analysis in the graph, the calculation amount implemented in traversal class applications for the dataset through the three sorting methods is relatively large, especially on the BC algorithm, the calculation amount of each side is higher than that of BFS and SSSP algorithms.

As shown in Fig. 15, in the application of the BFS algorithm, the computational complexity of the CA dataset sorted by BAR algorithm is 14.71% lower than DBR algorithm, and 38.29% lower than Corder algorithm. The calculation amount of HU dataset sorted by BAR algorithm is 26.31% lower than DBR algorithm, and 27.36% lower than Corder algorithm.

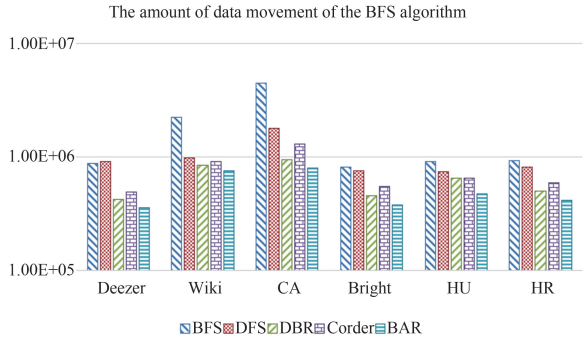


Fig. 15 The amount of computation for different vertex sorting methods on the BFS algorithm

As shown in Fig. 16, in the SSSP algorithm application, the calculation amount of the CA dataset sorted by the BAR algorithm decreases by 20.99% compared with DBR algorithm, and the calculation amount decreases by 44.75% compared with Corder algorithm. After sorting the Bright dataset using the BAR algorithm, the computational complexity is reduced by 32.56% compared with DBR algorithm, and by 43.47% compared with Corder algorithm.

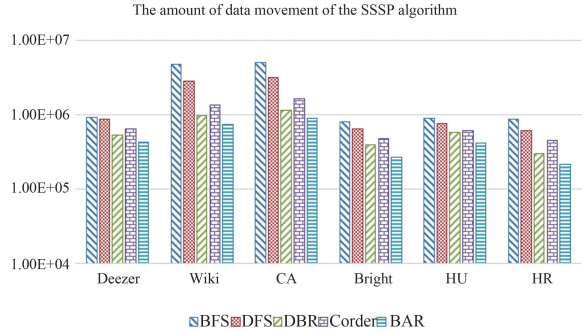


Fig. 16 The amount of computation for different vertex sorting methods on the SSSP algorithm

As shown in Fig. 17, in the application of BC algorithm, the calculation amount of Deezer dataset sorted by BAR algorithm is 21.98% lower than DBR al-

gorithm, and 44.96% lower than Corder algorithm. After sorting the HR dataset using the BAR algorithm, the computational complexity is reduced by 25.96% compared with DBR algorithm, and by 26.55% compared with Corder algorithm.

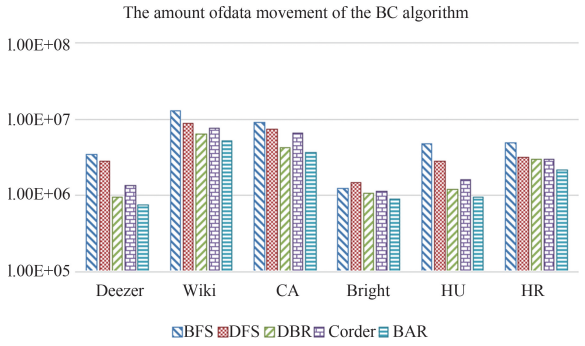


Fig. 17 The amount of computation for different vertex sorting methods on the BC algorithm

From the above analysis, it can be seen that BAR algorithm can effectively reduce the computational complexity. The maximum reduction is 32.56% compared with DBR algorithm and 44.96% compared with Corder algorithm.

Through the analysis of experimental results, in a few cases, the execution time of BAR algorithm is greater than Corder algorithm. The amount of data movement is higher than that of the Corder algorithm. However, it can be found that after the graph dataset is preprocessed by the BAR algorithm, it can effectively reduce the execution time, reduce the amount of data movement and calculation of the algorithm. Therefore, it can be concluded that the BAR algorithm can effectively alleviate the problem of poor locality of graph data in the process of graph calculation.

5 Conclusions

This paper focuses on the problem of poor locality of graph data in graph computation. Based on the idea of graph vertex reordering and the DBR algorithm, this paper proposes a BAR algorithm. And map the BAR algorithm onto a reconfigurable video array for acceleration. The experimental results show that compared with previous algorithms, BAR algorithm achieves significant improvements in performance such as execution time, data movement, and computational complexity, effectively alleviating the bottleneck caused by poor locality of graph data. However, the unstructured and irregular nature of large-scale graph data remains a huge challenge for graph computing. In future research work, the idea of reordering graph vertices remains an important approach to solving these problems. At the same time, the reconfigurable graph computing accelerator will also be one of the important directions to solve the bottleneck of graph computing.

References

[1] SAKR S, BONIFATI A, VOIGT H, et al. The future is big graphs: a community view on graph processing systems [J]. Communications of the ACM, 2021, 64(9): 62-71.

[2] WANG X, LIU K, LU W, et al. A fast cycle detection method for power grids based on graph processing [J]. CSEE Journal of Power and Energy Systems, 2023(9): 2204-2213.

[3] MUKKARA A, BECKMANN N, ABEYDEERA M, et al. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling [C]//2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Fukuoka, Japan: IEEE, 2018: 1-14.

[4] LEE J, AMORNPAISANNON B, MITRA T, et al. GraphWave: a highly-parallel compute-at-memory graph processing accelerator[C]//2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). Antwerp, Belgium: IEEE, 2022: 256-261.

[5] AWADELKARIM A, UGANDER J. Prioritized restreaming algorithms for balanced graph partitioning [C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, USA: Association for Computing Machinery, 2020: 1877-1887.

[6] SINGHANIA S, SHARMA N, VENKITARAMAN V, et al. CAR: community aware graph reordering for efficient cache utilization in graph analytics[C]//VLSI Design and Test: 26th International Symposium. Jammu, India: Springer, 2022: 453-467.

[7] GREEN O. Inverse-deletion BFS-revisiting static graph BFS traversals with dynamic graph operations [C]//2021 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, USA: IEEE, 2021: 1-7.

[8] CHAKRABORTY S, ENGELS C. Lower bounds for lexicographical DFS data structures [C]//2022 Data Compression Conference (DCC). Snowbird, USA: IEEE, 2022: 449-449.

[9] JIANG L, FENG R, WANG J, et al. DBR: a depth-branch-resorting algorithm for locality exploration in graph processing[C]//2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). Chiang Mai, Thailand: IEEE, 2022: 178-184.

[10] XIAO Y, SHAN R, YE Z, et al. APR-16: the physical design of a reconfigurable array processor chip[C]//2022 7th International Conference on Integrated Circuits and Microsystems (ICICM). Xi'an, China: IEEE, 2022: 422-429.

[11] NAI L, XIA Y, TANASE I G, et al. GraphBIG: understanding graph computing in the context of industrial solutions[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Austin, USA: IEEE, 2015: 1-12.

[12] KUMAR P, HUANG H H. Graphone: a data store for real-time analytics on evolving graphs[J]. ACM Transactions on Storage (TOS), 2020, 15(4): 1-40.

[13] GUI C Y, ZHENG L, HE B, et al. A survey on graph processing accelerators: challenges and opportunities[J].

Journal of Computer Science and Technology, 2019, 34: 339-371.
[14] LAKHOTIA K, SINGAPURA S, KANNAN R, et al. Recall: reordered cache aware locality based graph processing[C]// 2017 IEEE 24th International Conference on High Performance Computing (HiPC). Jaipur, India: IEEE, 2017: 273-282.
[15] BALAJI V, LUCIA B. Combining data duplication and graph reordering to accelerate parallel graph processing [C]//Proceedings of the 28th International symposium on high-performance parallel and distributed computing. New York, USA: Association for Computing Machinery, 2019: 133-144.
[16] ESFAHANI M K, KILPATRICK P, VANDIERENDONCK H. Locality analysis of graph reordering algorithms [C]//2021 IEEE International Symposium on Workload Characterization (IISWC). Storrs, USA: IEEE, 2021: 101-112.
[17] CHEN Y A, CHUNG Y C. Workload balancing via graph reordering on multicore systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(5): 1231-1245.
[18] CHEN Y A, CHUNG Y C. Workload balancing via graph reordering on multicore systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(5): 1231-1245.
[19] DENG J, JIANG L, ZHU Y, et al. HRM: H-tree based reconfiguration mechanism in reconfigurable homogeneous PE array[J]. Journal of Semiconductors, 2020, 41(2): 45-53.
[20] LESKOVEC J, KREVL A. SNAP datasets: stanford large network dataset collection [EB/OL]. (2014-06-30) [2023-05-06]. http://snap.stanford.edu/data.
[21] ROSSI R, AHMED N. The network data repository with interactive graph analytics and visualization [C]//Proceedings of the AAAI Conference on Artificial Intelligence. Austin, USA: AAAI Press, 2015: 1-6.

DENG Junyong, born in 1981. He received his Ph. D degree in system design of integrated circuit, at School of Microelectronics, in Xidian University in 2015. He visited the Laboratory of Computer Architecture of ECE, the University of Texas at Austin, as a post-doctoral researcher for one year and a half. His current research interests include ASIC design, reconfigurable computing, graph processing accelerator design, and performance evaluation of high-performance computing systems.