International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 14, No. 3, November 2025, pp. 843∼854
ISSN: 2089-4864, DOI: 10.11591/ijres.v14.i3.pp843-854

Parallel graph algorithms on a RISCV-based many-core

Ashuthosh Moolemajalu Ravikumar[1], Aakarsh Vinay[1], Krishna K. Nagar[2], Madhura Purnaprajna[1]
[1]Department of Electronics and Communications Engineering, PES University, Bengaluru, India
[2]Google, San Francisco, United States of America

Article Info

Article history:

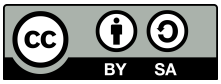
Received May 13, 2025
Revised Jul 17, 2025
Accepted Aug 7, 2025

Keywords:

Analytical model
Graph algorithm
Parallel architecture
Performance model
Reduced instruction set computer–five many-core

ABSTRACT

Graph algorithms are essential in domains like social network analysis, web search, and bioinformatics. Their execution on modern hardware is vital due to the growing size and complexity of graphs. Traditional multi-core systems struggle with irregular memory access patterns in graph workloads. Reduced instruction set computer–five (RISC-V)-based many-core processors offer a promising alternative with their customizable open-source architecture suitable for optimization. This work focuses on parallelizing graph algorithms like breadth-first search (BFS) and PageRank (PR) on RISC-V many-core systems. We evaluated performance based on graph structure and processor architecture, and developed an analytical model to predict execution time. The model incorporates the unique characteristics of the RISC-V architecture and the types and numbers of instructions executed by multiple cores, with a maximum prediction error of 11%. Our experiments show a speedup of up to $11.55\times$ for BFS and $7.56\times$ for PR using 16 and 8 cores, respectively, over single-core performance. Comparisons with existing graph processing frameworks demonstrate that RISC-V systems can deliver up to $20\times$ better energy efficiency on real-world graphs from the network repository.



Corresponding Author:

Ashuthosh Moolemajalu Ravikumar
Department of Electronics and Communications Engineering, PES University
Bengaluru, Karnataka, India
Email: ashuthoshmr@pes.edu

1. INTRODUCTION

Graph algorithms have become increasingly important in applications such as social networks, navigation, and graph neural networks. The methods for processing graphs have evolved significantly, transitioning from classical problems to modern general-purpose platforms like central processing units (CPUs) and graphics processing units (GPUs). Recent advancements include algorithmic improvements [1] and the development of specialized accelerators and frameworks, such as GraphBLAST [2] and GraphLily [3], which leverage GPUs and field-programmable gate arrays (FPGAs) for enhanced performance.

However, these advancements also highlight the unique challenges inherent to graph processing, particularly in achieving efficient parallelization. Graph properties, such as connectivity between vertices, vary widely depending on the graph type, leading to irregular memory access patterns. Additionally, many graph algorithms are memory-bound, making efficient memory utilization critical.

A prime example of application-architecture co-design is Google's TPU [4], which demonstrates a shift towards application-optimized parallel architectures to achieve greater efficiency. Similarly, custom accelerators targeting graph applications have been implemented on FPGAs [3], [5]. But the significant design time and effort required for FPGA-based accelerators often limit their practicality. GPUs [2] and vector pro-

cessors have emerged as alternative parallel architectures for graph application acceleration.

In this context, reduced instruction set computer–five (RISC-V-based) many-core processors have gained prominence due to their customizable and open-source architecture. However, there is still a need to understand how well graph workloads perform on RISC-V-based many-core processors. Our research aims to address this question by exploring graph algorithms and developing an analytical model to predict the performance of RISC-V-based many-core processors. This model will allow us to project the performance of parallel graph processing for core counts up to 16 and various graph types. Our contributions in this work include the following:

- Parallelizing graph algorithms on RISC-V-based many-core processors.
- Develop an analytical model to predict the performance of parallel graph algorithms on RISC-V-based many-core.
- Validate the accuracy of the model on real world graphs.

2. PARALLELIZING GRAPH ALGORITHMS ON RISC-V MANY-CORE

Among various graph algorithms, this work focuses on two widely applicable ones: breadth-first search (BFS) and PageRank (PR). Parallelizing these algorithms on a RISCV-based many-core involves distributing the workload across available cores. In this section, we examine the effects of parallelizing BFS with respect to the number of cores and graph types, which motivates the development of the analytical model. Our analysis leverages a RISC-V many-core called Mempool [6], featuring 16 cores and 64 KB of tightly coupled memory.

2.1. Parallel breadth-first search

BFS is a recursive algorithm that tries to traverse all the vertices in the graph starting from a single vertex known as the source or root vertex. Traversal involves exploring the neighbors of the vertex and updating the properties of the neighbors such as visited status, parent vertex and distance from the root vertex. The traversal of active vertices is done in every iteration and continues until there are no active vertices. Traversal can be done mainly in two ways, top-down approach and bottom-up approach. Based on the number of active vertices every iteration, certain approach is beneficial [1]. In our work, we consider bottom-up BFS as considerable time is spent on this kernel compared to the top-down BFS. As shown in Algorithm 1, in each iteration, all vertices (line 1) are checked to determine if they have not been visited (line 2). The neighbors of these unvisited vertices are then explored (line 3). If an incoming neighbor of an unvisited vertex is active (line 4), the unvisited vertex is marked as visited and activated for the next iteration (lines 5 and 6). Parallelizing this kernel across N cores involves dividing the total vertices (line 1) into N parts. While this division ensures equal distribution of vertices, the unvisited vertices (line 2) and their neighbors (line 3) are unevenly distributed. In graphs with highly irregular neighbor distributions (e.g., power-law graphs), this irregularity impacts performance by leaving some cores idle while others remain active. Increasing the number of cores in such cases will amplify the impact as very few cores compute the vertices with large number of neighbors.

Algorithm 1 . Single iteration of Bottom-up BFS

```
1:  for i = 1 to v do
2:      if cost[i] < 0 then
3:          for each j in incoming_neighbours of i do
4:              if active[j] > 0 then
5:                  active_list ← i
6:                  cost[i] ← iteration
7:                  break
8:              end if
9:          end for
10:     end if
11: end for
12: iteration ← iteration + 1
```

2.1.1. Graph type

Figure 1 presents the profile of the Mempool, showing the breakdown (%) of compute and stall times as a function of graph type. The analyzed graphs are real-world graphs listed in Table 1, sourced from the network repository [7]. For graphs with power-law characteristics, such as soc-wiki and web-polblogs, synchronization overhead contributes to stall times. This irregularity in the graph structure results in an uneven distribution of executed instructions across cores in the parallel many-core system.

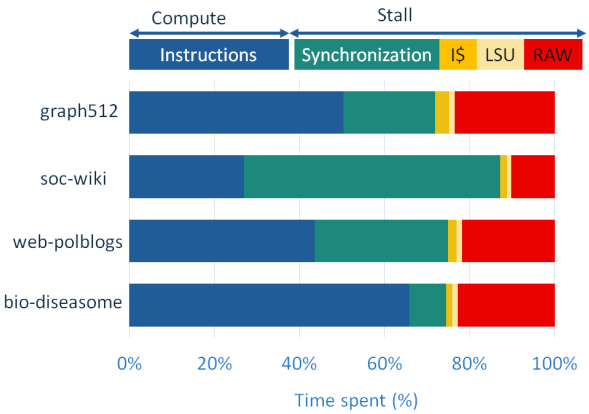


Figure 1. Percentage of time spent on compute and stalls for different types of graphs on a 16-core Mempool. Compute time includes instruction execution, while stalls are caused by synchronization, instruction cache misses, load-store unit (LSU) delays, and read-after-write (RAW) hazards

Table 1. Diverse type of graphs used in this work are from network repository [7]

Data	Vertices	Edges	Type
graph512	512	3114	Synthetic
soc-wiki	889	5828	Social network
web-polblogs	643	4560	Web network
bio-diseasome	516	2376	Biological network

2.1.2. Number of cores

Increasing the number of cores reduces latency but also introduces synchronization overhead. Figure 2 illustrates the latency breakdown for BFS traversal on soc-wiki with varying core counts. A maximum speedup of $9.2\times$ is achieved with 16 cores compared to single-core latency. Using 8 cores provides a $5.6\times$ speedup, while doubling the cores to 16 adds only an additional $1.6\times$ speedup.

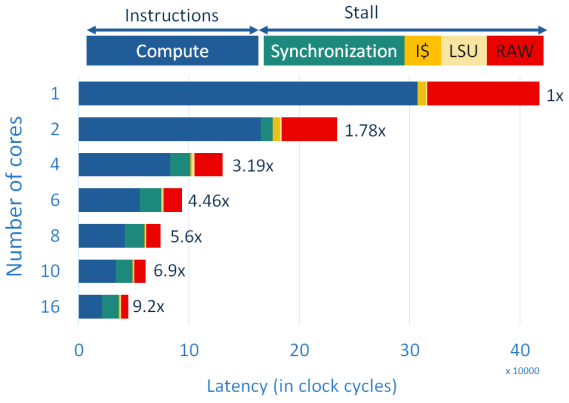


Figure 2. Latency breakdown (in clock cycles) for BFS traversal of the soc-wiki graph across various core counts

2.2. Need for performance prediction

In section 2.1., we examined how factors such as graph input type and core count influence the performance of graph algorithms on the parallel RISC-V architecture. However, accurately predicting performance prior to execution remains a challenge. To address this, we propose an analytical model that estimates cycle counts based on graph characteristics and system configuration. This model enables performance prediction for any given graph input on Mempool with N cores, without requiring execution on the actual hardware.

3. BUILDING THE ANALYTICAL MODEL: METHOD

Figure 3 shows the method of building an analytical model to predict the performance of graph algorithms. As shown in the Figure 3, the graph algorithm gets translated to set of RISC-V instructions by the compiler. These instructions include compute and memory instructions that is mapped to N cores.

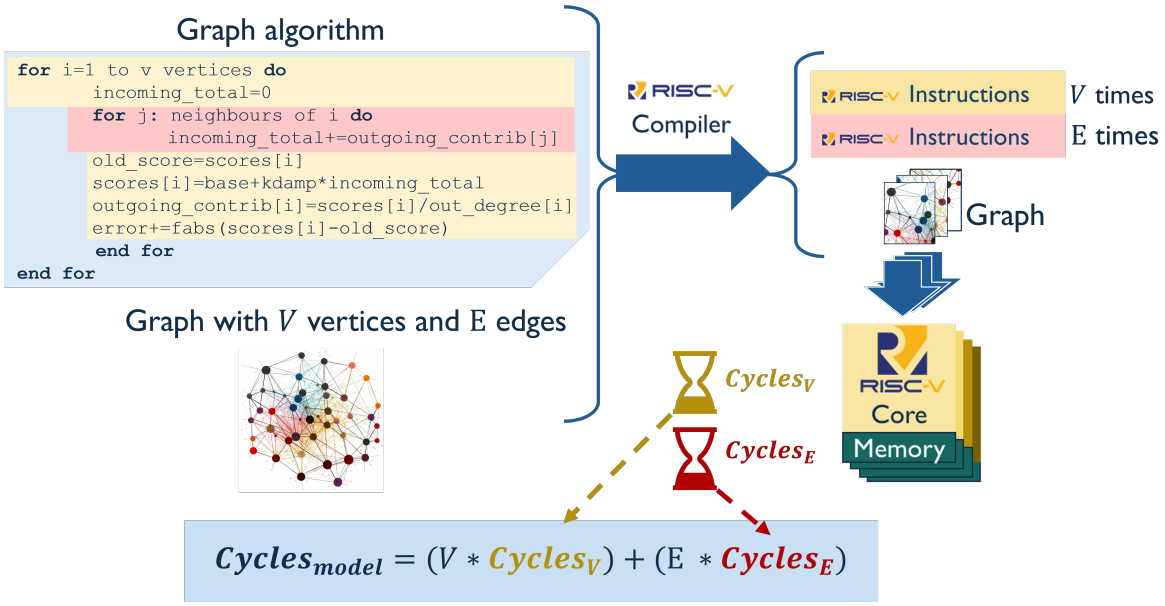


Figure 3. Illustration of the analytical model for predicting the performance of graph algorithms on a RISC-V many-core processor cluster

Along with the instructions, the graph is also distributed to the multi-core thereby dividing the overall computations. Based on the number and type of RISC-V instructions, the latency to execute the instructions vary and is given by $Cycles_V$ and $Cycles_E$. Similarly, these set of instructions execute depending on the graph properties such as, number of vertices (V) and edges (E) of the graph. Using these information, we build the analytical model based on graph properties and number and type of RISC-V instructions on N cores.

We have developed an analytical model to predict the performance of BFS and PR. The notations used in this model for BFS and PR are given in the Tables 2 and 3. The value of the notation dependent on core count of many-core, instruction set architecture (ISA), and graph input are also indicated in the tables.

3.1. Breadth-first search

As shown in the Algorithm 1, the lines 1, 2 are executed for all the vertices in every iteration. Of all the vertices, unvisited vertices $V_{unvisited}$ are explored (line 3). If the $incoming\_neighbours$ of the $V_{unvisited}$ are active i.e V_{active} (line 4) , the $unvisited$ vertices are put in the new active list i.e V_{update}(line 5, 6, and 7). Based on the number of V_{active}, $V_{unvisited}$, $V_{neighbours}$, the number and type of instructions executed varies. The total cycles for each iteration of the bottom-up BFS is given using the (1). Using this equation, latency for bottom-up BFS can be predicted ahead in time based on the number of vertices and edges provided V_{active}, $V_{unvisited}$, $V_{neighbours}$ and V_{update} are known every iteration. Knowing the V_{active}, $V_{unvisited}$, $V_{neighbours}$, and V_{update} during compile time is only possible after running BFS once. Otherwise, assumptions can be made

by dividing the total number of vertices equally among each iteration. Extending this to multiple cores involves estimating the latency of the graph partition that takes the longest latency. Since the amount of work done by each core is dependent on the size of the partition, we consider the partition with the largest vertices and edges for our estimation V_{max} and E_{max}.

$$Cycles_{\text{bottom\_bfs}} = \sum_{i=1}^{\text{Iteration}} \left[\left(V_{\text{active}} \times Cycles_{\text{active}} \right) + \left(V_{\text{unvisited}} \times Cycles_{\text{unvisited}} \right) + \left(V_{\text{neighbours}} \times Cycles_{\text{neighbours}} \right) + \left(V_{\text{update}} \times Cycles_{\text{update}} \right) \right] \quad (1)$$

Table 2. Notations used in the model for bottom-up BFS and its dependencies

Notation	Description	Dependence
V_{active}	Number of active vertices in a given iteration	Input graph
$V_{unvisited}$	Number of unvisited vertices traversed	Input graph
$V_{neighbours}$	Number of neighbouring vertices explored	Input graph
V_{update}	Number of new active vertices found in an iteration	Input graph
$Cycles_{active}$	Cycles to explore each V_{active} vertices	Multi-core architecture, ISA
$Cycles_{unvisited}$	Cycles to traverse each $V_{unvisited}$	Many-core architecture, ISA
$Cycles_{neighbours}$	Cycles to check each $V_{unvisited}$	Many-core architecture, ISA
$Cycles_{update}$	Cycles to put each unvisited vertex to active list	Many-core architecture, ISA

Table 3. Notations used in the model for PR and its dependencies

Notation	Description	Dependence
N	Number of cores	Multi-core architecture
V_{max}	Maximum number of vertices in a partition	Input graph
E_{max}	Maximum number of edges in a partition	Input graph
$Cycles_v$	Cycles to operate on vertices	Multi-core architecture, ISA
$Cycles_e$	Cycles to operate on edges	Multi-core architecture, ISA

3.2. PageRank

PR involves ranking the vertices of the graph based on the connectivity. Each vertex is initialised with a score based on the number of outgoing edges (i.e. degree of the vertex) called outgoing contribution. As shown in the pseudo-code Algorithm 2, the outgoing contribution is accumulated into $incoming\_total$ for each of the vertices (lines 3,4). Based on the $base\_score$ and $incoming\_total$, a new score is calculated (line 7). This process is repeated until the difference between the old score and the new score i.e the error (line 8) is less than the threshold limit or until the maximum iteration. Since each vertex's score can be estimated in parallel, PR can be accelerated by leveraging the cores present in the many-core system.

Algorithm 2 . Single iteration of PageRank

1: **for** $i = 1$ to v vertices **do**
2: $incoming\_total \leftarrow 0$
3: **for** $j : neighbours$ of i **do**
4: $incoming\_total \leftarrow incoming\_total + outgoing\_contribution[j]$
5: **end for**
6: $old\_score \leftarrow scores[i]$
7: $scores[i] \leftarrow base\_score + kdamp * incoming\_total$
8: $error+ = fabs(scores[i] - old\_score)$
9: $outgoing\_contrib[i] = scores[i]/out\_degree[i]$
10: **end for**

For a single-core design, the latency to calculate the ranks of the graph $G(V, E)$ with V number of vertices and E number of edges in each iteration includes reading and accumulating the outgoing contributions of each vertex and updating the score (line 2,3 of Algorithm 2). Reading outgoing contributions and accumulating for each vertex depends on the total number of edges E in the graph. Updating the scores of the vertex

based on the accumulated contributions is dependent on the number of vertices V (line 6 to 9 of Algorithm 2). The cycles for calculating the ranks of every iteration of the vertices of $G(V, E)$ can be formulated as (2):

$$Cycles_{PageRank} = [V \times Cycles_v] + [E \times Cycles_e] \quad (2)$$

Similarly, extending this to multiple cores involves estimating the latency of the group of vertices that takes the longest. The cycles for calculating the ranks of every iteration in the multi-core system can be formulated as (3):

$$Cycles_{PageRank} = [V_{max} \times Cycles_v] + [E_{max} \times Cycles_e] \quad (3)$$

Since in every iteration, the same set of operations are carried out, we assume total latency for multiple iterations can be obtained by multiplying the number of iterations with $Cycles_{PageRank}$. In (2) and (3) provide a simplified view of performance as a function of vertices and edges for PR.

4. EXPERIMENTAL SETUP

We evaluated the performance of our analytical models with the actual latency measured in terms of clock cycles. We used a variety of graphs ranging from synthetic to real-world social and biological networks as shown in the Table 1. The graphs in Table 1 are stored in the compressed sparse row (CSR) compression format. The focus of the experiments are to evaluate the model's predictive accuracy across different processor core counts and graph types.

4.1. Target reduced instruction set computer–five architecture

Mempool [6] is a RISCV-based many-core cluster with tightly coupled $L1$ shared memory that can be scaled upto 256 cores supporting $RV32IMAXpulpimg$ instructions. The hierarchical interconnect is responsible for a maximum latency of 5 cycles in the conflict-free access to shared memory. As shown in Figure 4, the default configuration consists of 4 groups, 16 tiles per group, 4 cores per tile. Total shared memory of 1MB is available where the memory is divided into sequential addressed space and interleaved addressed space. The size of the addressing is configurable.

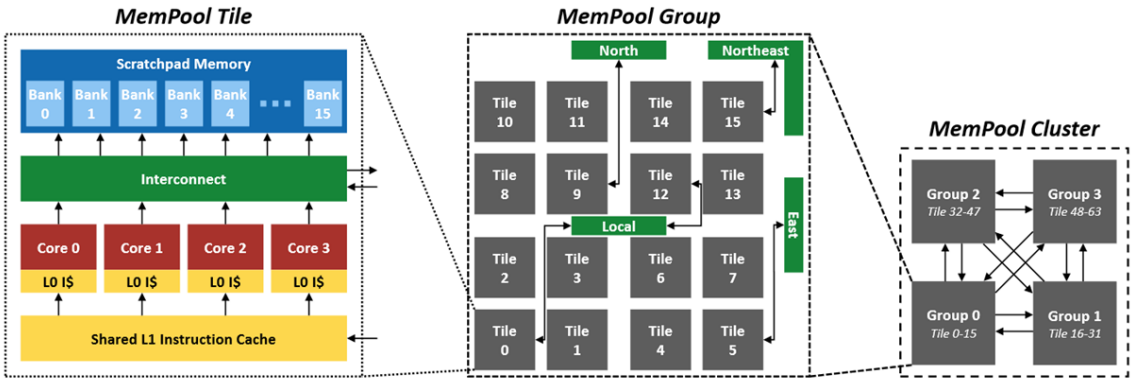


Figure 4. Many-core architecture of Mempool

For our experiments, we have used the Minpool configuration with 4 groups, 1 tiles per group and 4 cores per tile. This configuration has a total of 16 cores and 64 KB of tightly coupled data memory of which each tile has 8 KB of sequential addressed memory and 8 KB of interleaved addressed memory.

We conducted cycle-accurate RTL simulations [8] to obtain latency. For Mempool architecture, we measured $Cycles_{active}$, $Cycles_{unvisited}$, $Cycles_{neighbors}$, and $Cycles_{update}$ by reading the timing register which remains fixed for Mempool architecture. Using (1), and assuming the V_{part} and $degree$ due to the dynamic nature of the BFS, we predict the latency. Similarly, we obtain the exact number of active vertices in each iteration and the edges explored by running the bottom-up BFS once. We plug these numbers into (1) and calculate the latency which indicates static nature for a given graph.

Since PR required float support, we utilized a similar setup with an 8-core cluster called Snitch [9] with low-latency memory of 128 KB. We measured architecture dependent $Cycles_v$ and $Cycles_e$ cycles by reading the timing register of Snitch cluster for processing each vertex and edge. Using these values, we plugged the corresponding V_{max} and E_{max} of the partitions in consideration as shown in the (3).

5. RESULTS

In this section, we validate our analytical model and showcase the accuracy of our analytical model compared to the cycle-accurate simulation. We compare our work with existing state-of-the-art graph processing frameworks.

5.1. Accuracy of the analytical model

In this sub-section, we evaluate the performance of our analytical model in predicting the latency of BFS and PR.

5.1.1. Breadth-first search

Figure 5 shows the accuracy ranges of performance predictions made for core counts ranging from 1 to 16 for traversing BFS. We observe an accuracy in the range $47 - 94\%$. This accuracy is due to the dynamic nature of BFS where the number of active, visited and unvisited vertices are not known during compile time. Due to this dynamic nature assumptions we observe low predictive accuracy. The right hand side of the Figure 5 shows the accuracy of the predictions when the number of active, visited and unvisited vertices are known beforehand (by running BFS once). For this static case of BFS, the accuracy ranges from $88 - 99\%$. A maximum speedup of $11.55\times$ is observed on the 16-core compared to single-core for the traversal of the graph512.

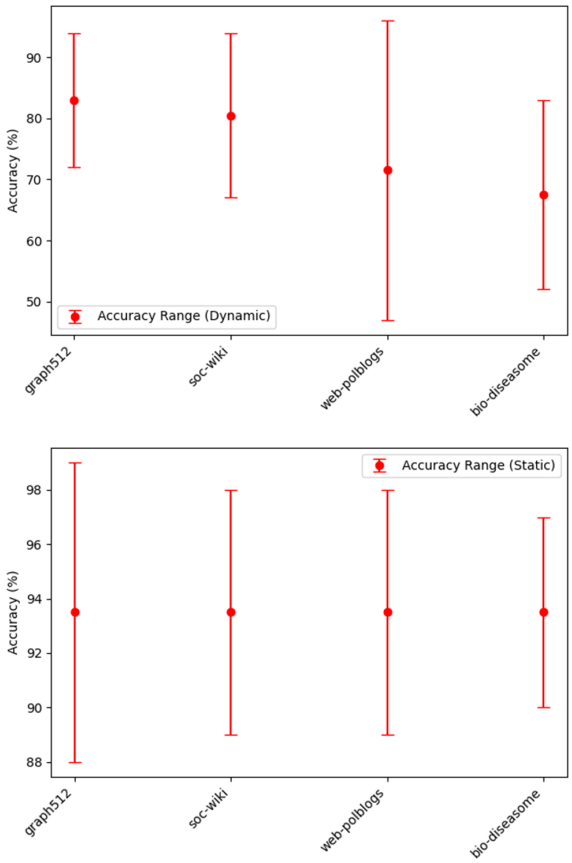


Figure 5. Comparison of performance prediction and actual latency obtained for Mempool with core count 1–16

5.1.2. PageRank

In Figure 6, we have compared the predicted latency and measured latency, and observe accuracy in the range of 93% upto 99%. Since PR involves updating all the scores in every iteration, the (3) provides an accurate performance prediction. Using our model, we can accurately predict the achievable speedup for any input graph and core counts up to 8. A maximum speedup of $7.56\times$ is observed on the 8-core compared to single-core for the graph512 input.

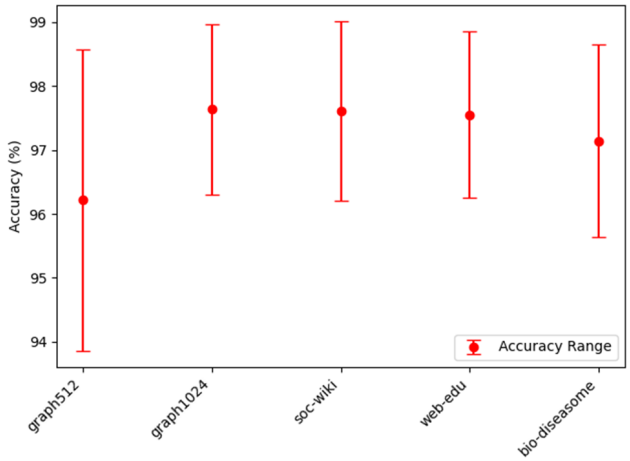


Figure 6. Comparison of performance prediction and actual latency obtained for Snitch's core count 1–8. Deterministic nature of algorithm leads to high accurate predictions

5.2. Comparison of million traversed edges per second of state-of-the-art graph processing frameworks

Table 4 compares state-of-the-art graph accelerators with RISC-V-based many-core. For comparison, we use the geometric mean of million traversed edges per second (MTEPS) and MTEPS per watt (MTEPS/W) as metrics. In our case, we evaluate an 8-core Snitch cluster implemented on an FPGA and on a 22 nm technology node, running both BFS and PR. For a fair comparison, all numbers correspond to bottom-up BFS and PR with graphs stored in CSR format. The MTEPS (geometric mean) and power values for comparison are sourced from experiments conducted in [3]. The GraphIt [10] results are based on a two-socket 32-core 2.8 GHz Intel Xeon Gold 6242 machine with 384 GB DDR4 memory and a bandwidth of 282 GB/s. GraphBLAST [2] experiments utilize a GTX 1080 Ti GPU with 3584 CUDA cores running at a peak frequency of 1582 MHz and 11 GB of GDDR5X memory, providing a bandwidth of 484 GB/s. The GraphLily [3] accelerator operates on a Xilinx Alveo U280 FPGA with a 165 MHz frequency and 16 HBM channels.

With GraphLily, the MTEPS is comparable to GraphBLAST but GraphLily achieves better MTEPS/W indicating higher efficiency than GraphBLAST. Although Snitch eight-core cluster provides low MTEPS, MTEPS/W is $20\times$ better than state-of-the-art highlighting the energy efficiency of RISCV-based many-core processors in graph processing.

Table 4. Comparison with state-of-the-art graph processing accelerators. The graphs considered in our work are very small and fit within the tightly coupled memory (128 KB) of the Snitch cluster

Related works	Technology (nm)	Algorithm (geometric mean)	MTEPS (W)	Power	MTEPS/W	Model
GraphIt [10] (CPU)	22	BFS	1957	264	7	No
		PageRank	2280	268	9	
GraphBLAST [2] (GPU)	16	BFS	4114	146	28	No
		PageRank	4940	182	27	
GraphLily [3] (FPGA)	16	BFS	3581	45	80	No
		PageRank	5591	49	114	
This work [9] (FPGA)	16	BFS	5	0.9	6	Yes
		PageRank	20	0.9	23	
This work [9]	22	BFS	103	0.17	605	Yes
		PageRank	386	0.17	2275	

5.3. Scalability analysis for core counts above 16

Figure 7 shows the performance scaling of BFS traversal on soc-wiki beyond 16 cores using a 256-core configuration of Mempool. Up to 16 cores, the analytical model predicts performance with an accuracy of 89%. However, beyond 16 cores, the accuracy drops because the model does not account for increasing synchronization delays and contention in shared resources. As core counts increase, these factors introduce additional latency, which impacts performance and accuracy of the model. At 256 cores, the accuracy drops to 21%. Improving the model to better capture these effects could enhance its accuracy at higher core counts.

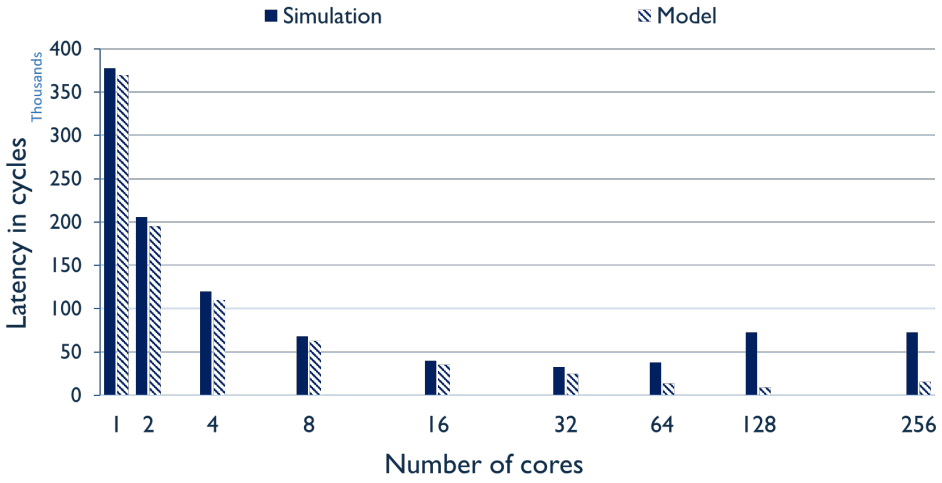


Figure 7. Comparison of performance predicted for BFS traversal of soc-wiki on core counts higher than 16, up to 256 cores

6. RELATED WORK

In this section, we review prior efforts in accelerating graph algorithms across various platforms, including many-core CPUs, GPUs, and FPGAs. We examine the usability of analytical models for performance prediction, highlighting their potential to guide architecture and algorithm design decisions.

6.1. Many-core, central processing unit-based accelerators

Eyerman *et al.* [11] explored the scalability of many-core architectures for graph processing using OpenMP-based implementations from the GAP benchmark suite [12]. While their work examined x86-based systems, our research targets low-power RISC-V architectures. Domain-specific languages (DSLs) such as GraphIt [10] have also been developed to map graph algorithms onto diverse platforms, including multi-core CPUs, GPUs, and RISC-V-based many-core systems.

6.2. Graphics processing unit-based accelerators

GPUs have also been used to accelerate graph applications. GraphBLAST [2] is a well known linear algebra GraphBLAS-based [13] framework. GraphBLAST implements partitioning schemes that align along vertex-centric partitioning or edge-centric partitioning.

6.3. Field-programmable gate array-based accelerators

Although FPGA-based accelerators require high design effort, often limiting their practicality, there have been efforts to accelerate graph processing using these platforms. Early frameworks like GraphGen [14] introduced a vertex-centric approach to utilize FPGAs for graph computations. FPGP [15] advanced this with an interval-shard structure, enabling flexibility across various graph algorithms without requiring complete reimplementation. ForeGraph [16] leveraged BRAM resources across multiple FPGA boards, while FabGraph [17] further optimized performance by introducing a two-level caching mechanism.

To address the challenges of high design complexity, recent frameworks have focused on simplifying implementation. HitGraph [18] used vertical partitioning to increase partition size, though preprocessing

overheads from edge sorting remained a limitation. AccuGraph [19] addressed data conflicts with a parallel conflict resolver but still relied on edge sorting during partitioning. High-level synthesis (HLS)-based solutions, such as ThunderGP [5], utilized the gather-apply-scatter (GAS) model to improve efficiency. Frameworks like GraphLily [3], designed for HBM-equipped FPGAs, and GraphSoC [20], which introduced custom graph instructions, further reduced design complexity and bitstream generation times.

6.4. Usability of performance models

GAIL [1] is one of the early works to analytically model graphs by examining the number of edges traversed, memory accesses, and access times. The graph algorithm iron law suggests that implementation or algorithm improvements are better evaluated empirically than analytically. However, our work uses an analytical approach to provide insights into implementation-specific improvements.

Chhugani *et al.* [21] developed a scalable BFS for modern CPUs, using dynamic partitioning based on cache size to improve traversal speed and locality. They also introduced a performance model that achieved $85 - 90\%$ accuracy in evaluating traversal phases. Verstraaten *et al.* [22] highlighted the importance of graph structure in determining the performance of different strategies during PR iterations. Similarly, our work considers graph structure as a key factor in performance analysis. Verstraaten *et al.* [23] also modeled graph application performance on GPUs using a data-driven approach. Although dynamic scheduling reduced analytical model accuracy to less than 50%, the use of large datasets and binary decision trees improved prediction accuracy. Their analysis helped identify the best implementation strategies for BFS traversal. Although such works accelerate and model graph applications on various parallel architectures, these works do not develop an analytical model that aids in coming up with an optimal design on the basis of graph properties.

7. CONCLUSION AND FUTURE WORK

In this work, we developed an analytical model resembling the RISCV-based many-core architecture for performance estimation of graph algorithms such as BFS and PR. This model predicts performance based on the graph structure, even before executing the workload on the many-core. By providing insights into the expected performance, the model enables informed decision-making regarding the optimal number of cores when deploying RISC-V many-cores on FPGAs for parallel graph processing. The adaptability of the model extends beyond RISC-V many-cores, as it can be tailored to other parallel many-core architectures by updating the architectural parameters. The analytical model shows a strong correlation with cycle-accurate RTL simulations of the RISCV-based many-core system, with a maximum prediction error of 11% observed on real-world graphs from the network repository across various core counts. The target RISC-V many-core architecture delivers comparable MTEPS/W on FPGAs and achieves up to 20x better MTEPS/W on a 22 nm technology node compared to an FPGA-based graph accelerator implemented on a 16 nm process.

Future work involves validating the versatility of the analytical model using different architectures, extending support for additional graph algorithms, and exploring higher core counts and larger graph sizes. This work highlights the potential of analytical modeling for guiding hardware design in graph processing.

ACKNOWLEDGMENTS

We thank Rajesh Vivekanandham and Venkatraman Ramakrishnan for their guidance and support as industry liaisons. Their input was helpful in aligning our work with industry perspectives.

FUNDING INFORMATION

This work is supported by Semiconductor Research Corporation (SRC) as part of Indian Research Program (Project ID 3055.001).

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Ashuthosh Moolemajalu Ravikumar	✓	✓	✓	✓	✓	✓		✓	✓	✓				
Aakarsh Vinay					✓	✓			✓	✓				
Krishna K. Nagar	✓									✓		✓	✓	
Madhura Purnaprajna	✓				✓	✓				✓	✓	✓	✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Derived data supporting the findings of this study are available from the corresponding author on request.

REFERENCES

[1] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, vol. 21, no. 3-4, pp. 137–148, 2012, doi: 10.3233/SPR-130370.

[2] C. Yang, A. Buluç, and J. D. Owens, "Graphblast: A high-performance linear algebra-based graph framework on the GPU," *ACM Transactions on Mathematical Software (TOMS)*, vol. 48, no. 1, pp. 1–41, Feb. 2022, doi: 10.1145/3466795.

[3] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "Graphlily: Accelerating graph linear algebra on hbm-equipped FPGAs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Munich, Germany, 2021, pp. 1-9, doi: 10.1109/ICCAD51958.2021.9643582.

[4] Google TPU, Google. [Online]. Available: https://cloud.google.com/tpu, (Accessed: Jan. 25, 2021).

[5] X. Chen, H. Tan, Y. Chen, B. He, W.-F. Wong, and D. Chen, "Thundergp: Hls-based graph processing framework on FPGAs," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 69–80, doi: 10.1145/3431920.3439290.

[6] M. Cavalcante, S. Riedel, A. Pullini, and L. Benini, "MemPool: A shared-L1 memory many-core cluster with a low-latency interconnect," in *2021 Design, Automation, and Test in Europe Conference and Exhibition (DATE)*, Grenoble, FR, Mar. 2021, pp. 701–706, doi: 10.23919/DATE51398.2021.9474087.

[7] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 29, no. 1, 2015, doi: 10.1609/aaai.v29i1.9277.

[8] Verilator, Veripool. [Online]. Available: https://veripool.org/verilator, (Accessed: May 21, 2022).

[9] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845-1860, Nov. 2021, doi: 10.1109/TC.2020.3027900.

[10] Y. Zhang, M. Yang, R. Baghdadi, S. Kamil, J. Shun, and S. Amarasinghe, "Graphit: A high-performance dsl for graph analytics," *arXiv preprint arXiv:1805.00923*, 2018.

[11] S. Eyerman, W. Heirman, K. D. Bois, J. B. Fryman, and I. Hur, "Many-core graph workload analysis," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 282–2922, doi: 10.1109/SC.2018.00025.

[12] S. Beamer, K. Asanović, and D. Patterson, "Gail: The graph algorithm iron law," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, 2015, pp. 1–4, doi: 10.1145/2833179.2833187.

[13] Graphblas. [Online]. Available: https://graphblas.org/, (Accessed: Apr. 20, 2022)

[14] E. Nurvitadhi *et al.*, "Graphgen: An FPGA framework for vertex-centric graph computation," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. Boston, MA, USA, 2014, pp. 25-28, doi: 10.1109/FCCM.2014.15.

[15] G. Dai, Y. Chi, Y. Wang, and H. Yang, "FPGP: Graph processing framework on FPGA a case study of breadth-first search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 105–110, doi: 10.1145/2847263.2847339.

[16] G. Dai, T. Huang, Y. Chi, N. Xu, Y. Wang, and H. Yang, "Foregraph: Exploring large-scale graph processing on multi-fpga architecture," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 217–226, doi: 10.1145/3020078.3021739.

[17] Z. Shao, R. Li, D. Hu, X. Liao, and H. Jin, "Improving performance of graph processing on fpga-dram platform by two-level

vertex caching," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 320–329, doi: 10.1145/3289602.3293900.
[18] S. Zhou, R. Kannan, V. K. Prasanna, G. Seetharaman, and Q. Wu, "Hitgraph: High-throughput graph processing framework on FPGA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249–2264, 2019, doi: 10.1109/TPDS.2019.2910068.
[19] P. Yao, L. Zheng, X. Liao, H. Jin, and B. He, "An efficient graph accelerator with parallel data conflict management," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–12, doi: 10.1145/3243176.3243201.
[20] N. Kapre, "Custom fpga-based soft-processors for sparse graph acceleration," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Toronto, ON, Canada, 2015, pp. 9–16, doi: 10.1109/ASAP.2015.7245698.
[21] J. Chhugani, N. Satish, C. Kim, J. Sewall, and P. Dubey, "Fast and efficient graph traversal algorithm for cpus: Maximizing single-node efficiency," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 378–389, doi: 10.1109/IPDPS.2012.43.
[22] M. Verstraaten, A. L. Varbanescu, and C. de Laat, "Quantifying the performance impact of graph structure on neighbour iteration strategies for pagerank," in *Euro-Par 2015: Parallel Processing Workshops*, 2015, pp. 528–540, doi: 10.1007/978-3-319-27308-2_43
[23] M. Verstraaten, A. L. Varbanescu, and C. de Laat, "Using graph properties to speed-up gpu-based graph traversal: A model-driven approach," *arXiv preprint arXiv:1708.01159*, 2017.

BIOGRAPHIES OF AUTHORS

Ashuthosh Moolemajalu Ravikumar has a B.Tech. in Electronics and Communication Engineering from PES University Bangalore. He is currently working on design and high-speed implementation of Error-correcting codes in high-performance communication systems. His research includes digital system design and signal processing. He can be contacted at email: ashuthoshmr@pes.edu.

Aakarsh Vinay received his bachelor's degree in Electronics and Communication Engineering from PES University, Bangalore in May 2024. He is currently working as a Project Associate at the Indian Institute of Science along with advanced micro devices (AMD). His research interests include computer architecture, hardware/software co-design, and heterogeneous systems. He can be contacted at email: aakarshvinay1030@gmail.com.

Krishna K. Nagar is a seasoned hardware design engineer and researcher focused on AI accelerator technology, CPU microarchitecture and NoC. He currently works at Google on TPU development. He has mentored several students on research projects. He did his Ph.D. from University of South Carolina and BE from RGPV, India. He can be contacted at email: kknagar@google.com.

Madhura Purnaprajna received her Master's in Electrical and Computer Engineering from University of Alberta, Canada in January 2005; and her Ph.D. in Electrical Engineering from the Heinz Nixdorf Institute, University of Paderborn, Germany in December 2009. Currently, she serves as Professor at the Department of Electronics and Communication, PES University, Bengaluru. Her research interests are in Reconfigurable Computing and Processor Architectures. She can be contacted at email: madhurap@pes.edu.