Accelerating PageRank Algorithmic Tasks with mIPU

Md Rownak Hossain Chowdhury, Mostafizur Rahman
Division of Energy, Matters and Systems, Uuniversity of Missouri-Kansas City (UMKC)
Kansas City, MO, US
{rhctmc, rahmanmo} @umkc.edu

***Abstract*— Addressing the growing demands of artificial intelligence (AI) and data analysis requires new computing approaches. In this paper, we propose a reconfigurable hardware accelerator designed specifically for AI and data-intensive applications. Our architecture features a messaging-based intelligent computing scheme that allows for dynamic programming at runtime using a minimal instruction set. To assess our hardware's effectiveness, we conducted a case study in TSMC 28nm technology node. The simulation-based study involved analyzing a protein network using the computationally demanding PageRank algorithm. The results demonstrate that our hardware can analyze a 5,000-node protein network in just 213.6 milliseconds over 100 iterations, all while consuming only 4.1mW of power. These outcomes signify the potential of our design to achieve cutting-edge performance in next-generation AI applications.**

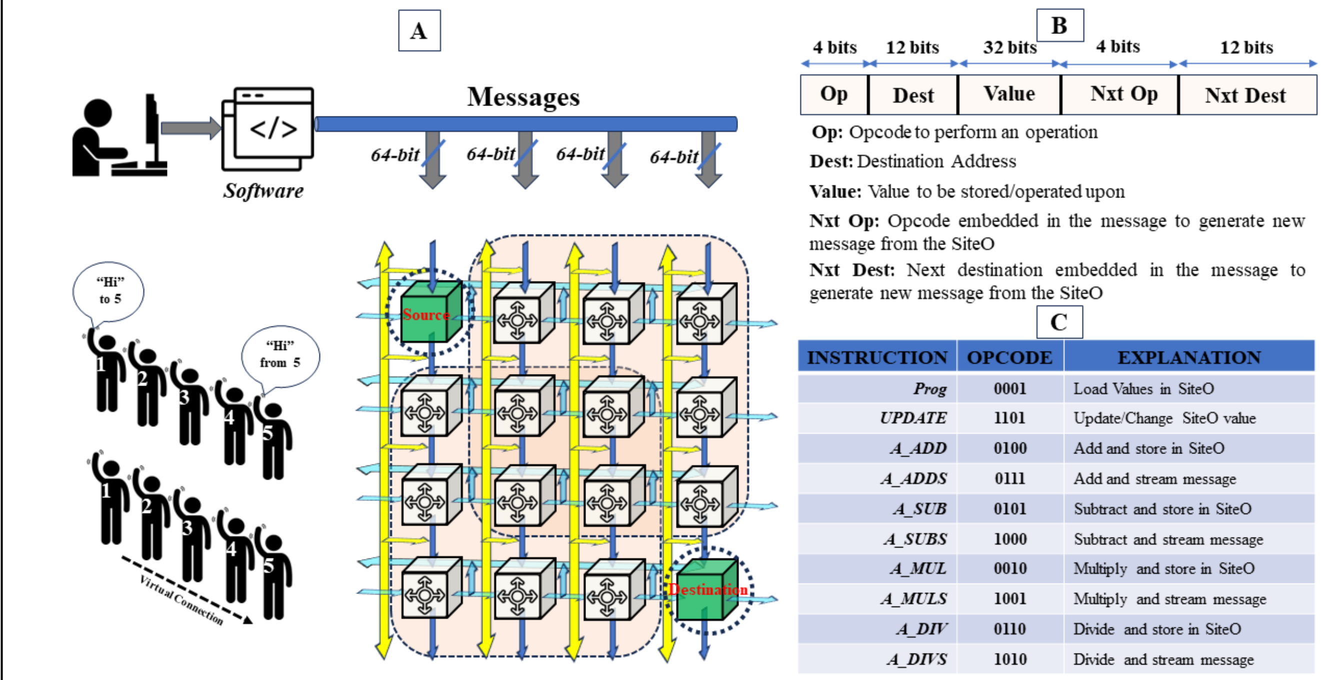
Keywords— Reconfigurable Computing, Hardware Accelerator, Artificial Intelligence, PageRank Algorithm

I. Introduction

The emergence of artificial intelligence and its integration in various scientific and engineering applications have led to an unprecedented surge in data generation across various domains such as bioinformatics, genomics and more. This immense data growth has the potential to unfold valuable insights to produce data-driven decisions and optimize existing process. However, data-intensive tasks suffer from various computational challenges to handle vast amount of data as manipulating such huge volume of data involves intricate algorithm and parallel processing [1]. Hence, developing sophisticated hardware architecture capable of handling massive data has become a center of attention among researchers in both industry and academia.

Considering the mammoth datasets and their underlying complex data mapping, modern data analytic tasks demand hardware architectures with agility, scalability, and adaptability without sacrificing performance. Consequently, several domain-specific hardware accelerators like SpArch [2], MatRaptor [3] have been proposed with improved dataflow mechanism; however, these designs do not provide inherent data processing capabilities that hinders their adaptability to variations in data size, model complexity, and software frameworks irrespective of applications. Again, architecture like MEISSA [4] offers low latency; hence, is beneficial in edge devices whereas TPU [5] provides high throughput and shows better performance in data centers.

Given the existing whitespace between the advancement in algorithm and the need for efficient hardware to adapt these algorithms, researchers are also focusing on improving computing technology. Several computing techniques like



Op: Opcode to perform an operation
Dest: Destination Address
Value: Value to be stored/operated upon
Nxt Op: Opcode embedded in the message to generate new message from the SiteO
Nxt Dest: Next destination embedded in the message to generate new message from the SiteO

INSTRUCTION	OPCODE	EXPLANATION
Prog	0001	Load Values in SiteO
UPDATE	1101	Update/Change SiteO value
A_ADD	0100	Add and store in SiteO
A_ADDS	0111	Add and stream message
A_SUB	0101	Subtract and store in SiteO
A_SUBS	1000	Subtract and stream message
A_MUL	0010	Multiply and store in SiteO
A_MULS	1001	Multiply and stream message
A_DIV	0110	Divide and store in SiteO
A_DIVS	1010	Divide and stream message

Figure 1 (A) Overview and concept of programmable hardware architecture (B) Message encoding scheme (C) Instruction Set Architecture (ISA).

fine-grained polymorphic circuit [6], noise-based configurable computing [7], crosstalk built-in memory [8] have been proposed to improve logic computations. Even though these techniques show prospects in circuit performance, there is still necessity to develop hardware that can be programmed at run-time for AI and data intensive tasks. This gap has motivated us to develop a hardware architecture that has potential to perform essential operations like matrix-vector multiplication with intelligent dataflow within minimum time steps.

In this paper, we propose a novel programmable computing technology that revolves around a unique flexible virtual interconnection scheme where any computing unit can be connected to another at run-time. To facilitate this, we developed a lightweight instruction set architecture (ISA) that enables transferring values from one processing element to another inherently without any additional data processing mechanism. Our intelligent processing element resembles GPUs and TPUs in terms of streaming, distribution, and parallel nature but are distinct in their interconnection mechanism and hence efficient data pipelining. Therefore, the hardware architecture can handle large amount of data with the same efficiency since the reconfiguration and computation scheme is not rigid for certain data sizes.

In short, the key contributions of our design are as follows:

- An innovative computer architecture concept to achieve programmability at run-time.
- A compact instruction set architecture that enables intuitive data flow mechanism within hardware elements without software intervention.
- A low-latency matrix-vector multiplication approach using reconfigurable hardware structure.
- Evaluation of our programmable accelerator on computation-intensive PageRank algorithm to analyze protein network with high throughput.

II. Hardware Architecture

A. *Programmability*

Our innovation is a programmable computing architecture that can be reconfigured at run-time to behave as a custom ASIC through interconnection flexibility and, as a result, information processing. An analogy of the proposed interconnect configurability is shown in Figure 1A (left). If we assume that 5 people are standing in a line, and they generate and pass messages from left to right in a circular manner (#1 sends to #2, #2 sends to #3, and the rightmost person, #5, sends its message to #1), then any message from anyone can be delivered to anyone in this human chain. For example, if #1 generates a message "Hi" intended for #5, it passes it to #2, and #2, #3, and #4 keep passing the same message to their right until destination #5 is reached. In this message passing scheme, #2, #3, and #4 form a virtual link between #1 and #5. This source, destination-based message passing scheme can be applied to computing cores as well as shown in Figure 1A(right). We organize the cores in rows and columns and connect them in a manner that messages can be communicated between any cores. The cores are very light and capable of performing only essential operations like programming and arithmetic. A message originating from any of the cores in a 16 Core configuration (4 rows and 4 columns- Figure 1A(right)), goes right or down depending on the destination. To configure our proposed hardware unit, only messages need to be sent to proper cores which in turn sets destination addresses, values, and operations through register writes. The message encoding scheme is shown in Figure 1B. A core is capable of both programming and arithmetic operations. The opcode values act as guides to distinguish between programming and operation. The instruction set architecture (***ISA***) of our programmable hardware accelerator is shown in Figure 1C, that comprises only ***10*** instructions. Among these, one instruction (***Prog***) is designated for data loading and the remaining nine instructions (***UPDATE, A_ADD, A_SUB, A_MUL, A_DIV, A_ADDS, A_SUBS, A_MULS, A_DIVS***) are allocated for various mathematical manipulation.

The programmability concept of our proposed hardware accelerator is illustrated in Figure 2. Here, we used only four instructions and focused solely on a single row of the entire

A

[0:3]	[4:15]	[16:47]	[48:51]	[52:63]	MESSAGE	BINARY REPRESENTATION OF MESSAGES				
Prog	0	1.1	A_ADD	3	*M-1*	0001	000000000000	00111111100011001100110011001101	0100	000000000011
Prog	1	1.2	A_ADD	3	*M-2*	0001	000000000001	00111111100110011001100110011010	0100	000000000011
Prog	2	1.3	UPDATE	3	*M-3*	0001	000000000010	00111111101001100110011001100110	1101	000000000011
A_MULS	0	1	0	0	*M-4*	1001	000000000000	00111111100000000000000000000000	0000	000000000000
A_MULS	1	2	0	0	*M-5*	1001	000000000001	01000000000000000000000000000000	0000	000000000000
A_MULS	2	3	0	0	*M-6*	1001	000000000010	01000000010000000000000000000000	0000	000000000000

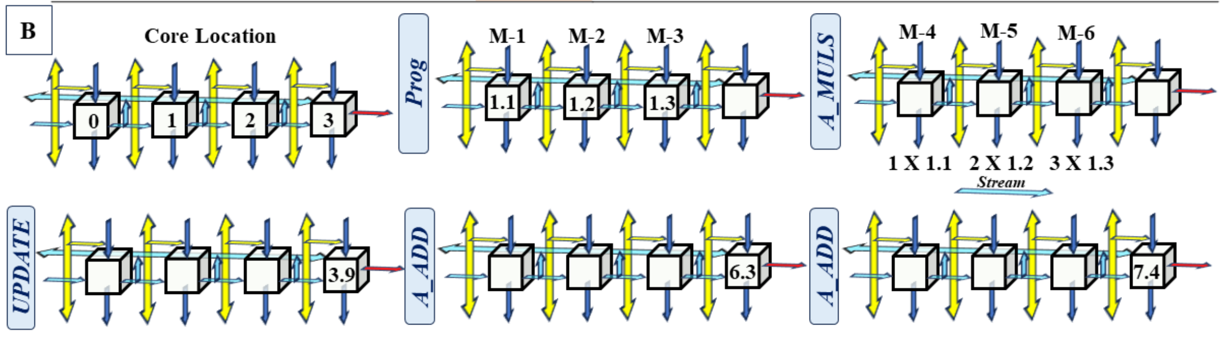


Figure 2(A) Bitwise segmentation of messages and (B) Illustration of the Concept of programmability utilizing hardware resources.

architecture for ease of demonstration. In this example, six messages shown in Figure 2A are provided as inputs to three cores (Core0, Core1, Core2) over two separate clock cycles. During the initial clock cycle, three messages (***M-1***, ***M-2***, and ***M-3***) are transmitted from the user end. As soon as, cores acknowledge these messages, the decoder unit of each core analyze the opcode (***from bit 0 to 3***) and the destination location (***from bit 4 to 15***). In this case, the opcode is "***Prog***" and hence the values (***1.1***, ***1.2***, ***1.3***) embedded in the message will be stored in the floating-point unit (FPU) of respective cores. Cores also retain the next opcode (***from bit 48 to 51***) and the next destination (***from bit 52 to 63***) integrated in the message. In the next clock cycle, the remaining three messages (***M-4***, ***M-5***, and ***M-6***) are sent. According to the opcode (***A_MULS***) of these messages, it first multiplies values (***1***, ***2***, ***3***) contained in the messages with the values (***1.1***, ***1.2***, ***1.3***) stored in the respective FPU. The opcode and destination are then updated according to the next opcode and next destination value stored in the core. Consequently, three multiplication results (***1.1***, ***2.4***, ***3.9***) will be streamed towards core3 with opcode "***A_ADD***", "***A_ADD***", and "***UPDATE***" respectively. Thus, core3 updates its value to ***3.9*** at first and then performs two consecutive addition operations. Finally, it stores ***7.9*** in core3.

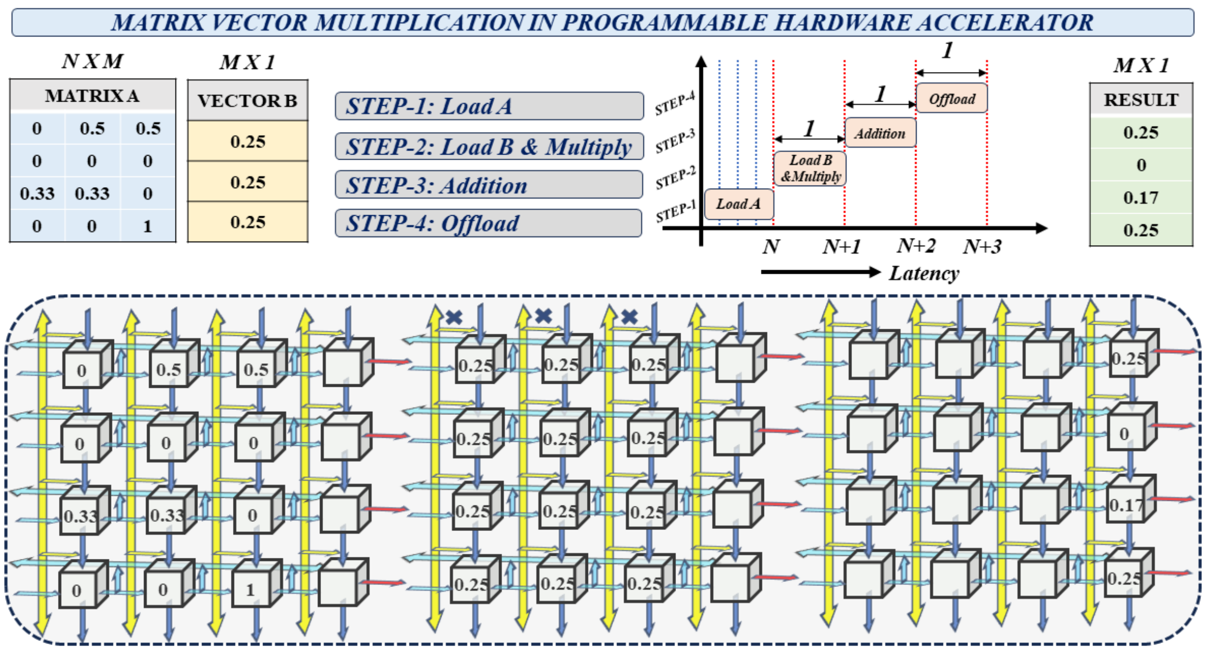


Figure 3 Matrix-Vector multiplication in our proposed programmable hardware accelerator.

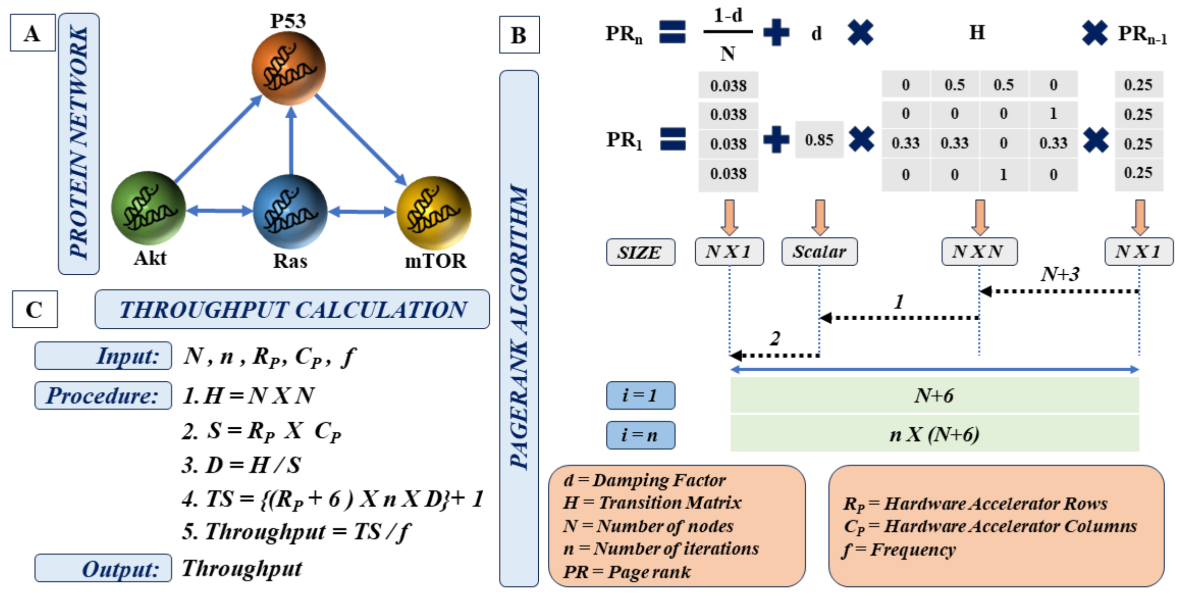
B. *Matrix-Vector Multiplication*

The matrix-vector multiplication operation using our proposed hardware unit can be divided into four (4) steps: ***1***) Data (***Matrix***) load ***2***) Data (***Vector***) load and multiply and ***3***) Addition and ***4***) offload. Figure 3 illustrates an example of the matrix-vector multiplication between ***matrix A*** (***4 X 3***) and ***vector B*** (***3 X 1***), showing the required time steps in each stage. Firstly, the entire ***matrix A*** is distributed across the fabric through hopping. The decoder unit of each core decodes the incoming message and programs each core with proper values. Upon loading ***matrix A***, the transpose of the ***vector B*** is transferred leveraging vertical bus, and multiplication operations are performed in each core. Afterward, the multiplication results are streamed to the desired location using horizontal bus, upgrading the message. Finally, the matrix-vector multiplication results are stored in the last core of each row of the hardware architecture after executing addition.

Suppose a matrix-vector multiplication takes two inputs: a matrix of size (***N X M***) and a vector of size (***M X 1***) then the number of cores required to store a matrix is (***N X M***). Since the vector has only ***1*** column, it will not occupy any additional cores. Additionally, we need another ***N*** number of cores to add the multiplication result. Thus, the number of cores necessary to perform a matrix-vector multiplication is ***{(N X M) + N}***. Here, the matrix will be loaded through hopping where each row of the matrix will be transferred one by one starting from the last row. Hence, the number of time steps required to load a matrix of size (***N X M***) is ***N***. Then, the vector will be sent using vertical bus and it will be multiplied with the matrix. This operation will require only ***1*** time step because of using vertical bus. Similarly, the addition operation will cost ***1*** time step utilizing horizontal bus. Finally, the result will be offloaded that will consume ***1*** more step. Thus, the total time steps required for the matrix-vector multiplication operation is (***N + 3***).

III. ACCELERATING PAGERANK ALGORITHM BASED ROTEIN NETWORK ANLYSIS LEVERAGING PROPOSED HARDWARE

Protein network is an indispensable tool for deciphering intricate molecular structures of biological processes; hence, analyzing protein networks is a fundamental task in biomedical research and drug discovery. However, existing protein network datasets are often incomplete and contain significant number of false positives [9]. Therefore, researchers are actively working on developing comprehensive protein network datasets like hu.MAP 2.0, HuRI, Y2H, Structure, DroRI etc [10-11]. Although large protein networks hold promise to perform rigorous inspection and drive innovation, choosing an apt algorithm to analyze such extensive network is also crucial. The famous

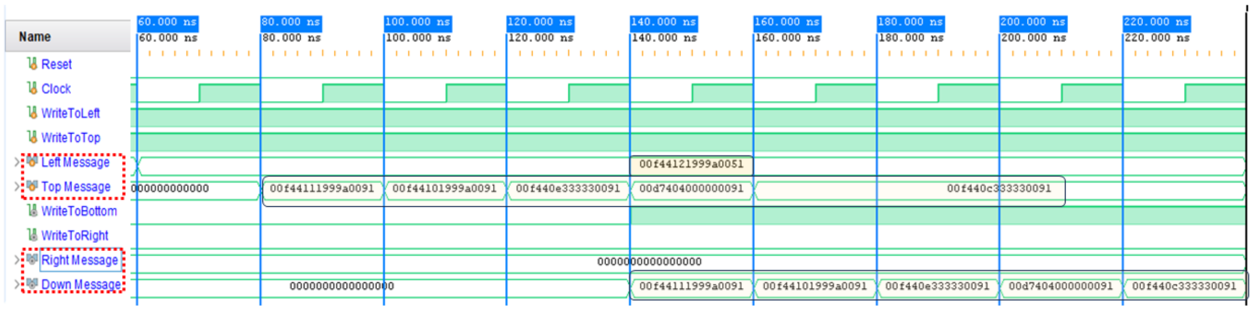

A
PROTEIN NETWORK
P53
Akt
Ras
mTOR
B
PAGERANK ALGORITHM
$PR_n = \frac{1-d}{N} + d \times H \times PR_{n-1}$
$PR_1 = \begin{bmatrix}0.038\\0.038\\0.038\\0.038\end{bmatrix} + 0.85 \times \begin{bmatrix}0 & 0.5 & 0.5 & 0\\0 & 0 & 0 & 1\\0.33 & 0.33 & 0 & 0.33\\0 & 0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix}$
SIZE
N X 1
Scalar
N X N
N X 1
N+3
1
2
i = 1
N+6
i = n
n X (N+6)
d = Damping Factor
H = Transition Matrix
N = Number of nodes
n = Number of iterations
PR = Page rank
R_P = Hardware Accelerator Rows
C_P = Hardware Accelerator Columns
f = Frequency
C
THROUGHPUT CALCULATION

```
Input:      N , n , R_P, C_P, f
Procedure:  1. H = N X N
            2. S = R_P X C_P
            3. D = H / S
            4. TS = {(R_P + 6 ) X n X D}+ 1
            5. Throughput = TS / f
Output:     Throughput
```

Figure 4(A) A small portion of large Protein network (B) Time steps required to implement the PageRank algorithm using our proposed hardware accelerator (C) Throughput calculation of the PageRank algorithm utilizing available resources.

PageRank algorithm [12], primarily used for Google search engine optimization algorithm, has been widely adapted to identify key insights about a protein network due to its ability to capture both global and local network information [13].

In protein network analysis applying PageRank algorithm, each protein is defined as nodes and the connections between proteins are referred to as edges. PageRank algorithm iteratively calculates the relative importance (page rank) of any protein (node) in the network following the equation shown in 4(B). Therefore, interpreting data-intensive protein network using PageRank algorithm puts forward various computational challenges to handle vast amount of data as manipulating such large datasets involves multiple iterations and computation expensive operation like matrix-vector multiplication. In such scenario, our programmable hardware architecture offers a convenient solution by accelerating the matrix-vector multiplication process.



MESSAGE	VALUE (Hex)	[0:3]	[4:15]	[16:47]	[48:51]	[52:63]	EXPECTATION	SIMULATION
LEFT-1	00f44121999a0051	Prog	5	10.1	A_ADD	15	*Decoded in SiteO 5*	✓
TOP-1	00f44111999a0091	Prog	9	9.1	A_ADD	15	*Pass to SiteO9 (Bottom)*	✓
TOP-2	00f44101999a0091	Prog	9	8.1	A_ADD	15	*Pass to SiteO9 (Bottom)*	✓
TOP-3	00f440e333330091	Prog	9	7.1	A_ADD	15	*Pass to SiteO9 (Bottom)*	✓
TOP-4	00d7404000000091	Prog	9	3	A_ADDS	13	*Pass to SiteO9 (Bottom)*	✓
TOP-5	00f440c333330091	Prog	9	6.1	A_ADD	15	*Pass to SiteO9 (Bottom)*	✓

Figure 5 Simulation result of our proposed reconfigurable hardware architecture.

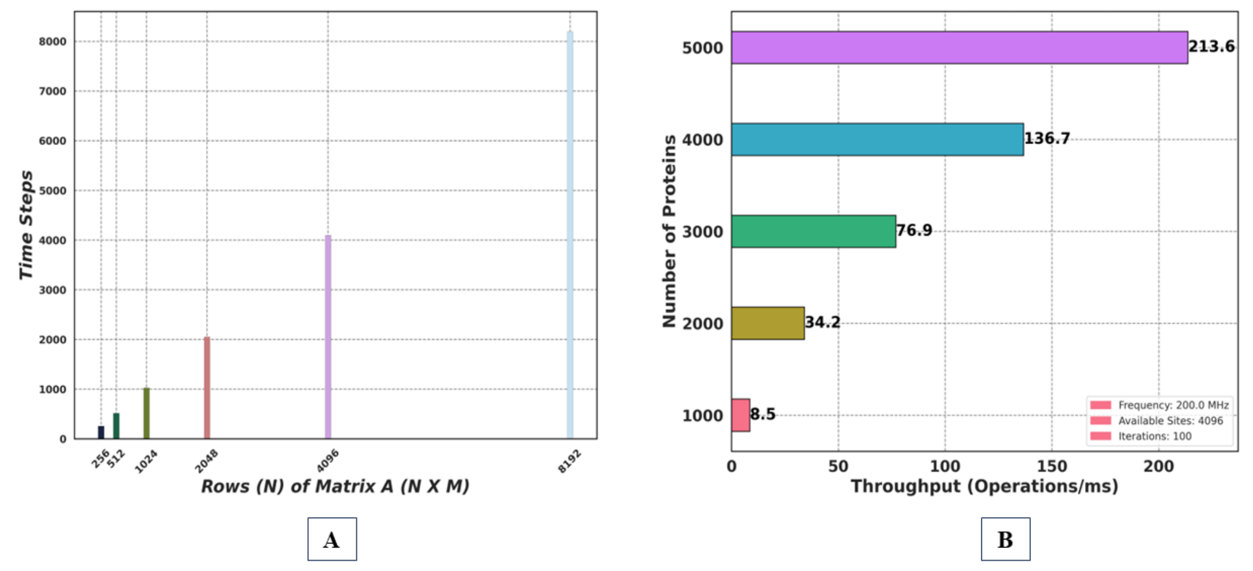


Figure 6(A) Latency calculation of Matrix-Vector multiplication varying rows of Matrix A (B) Throughput calculation varying the number of proteins in protein network.

To showcase the performance of our programmable hardware design, we have segmented the PageRank algorithm in several stages and calculated the required time steps in every stage over many iterations. Figure 5(B) demonstrates the total time steps required to complete one iteration of the PageRank algorithm. Here, the multiplication between the matrix (H) and the vector (PRn-1) requires N+3 steps; then, a scalar (d) will be loaded to be multiplied with the result of matrix vector multiplication that costs 1 time step. After that, the addition and offloading require 2 more steps. This process will be repeated several times based on the desired accuracy. Thus, the total time steps necessary for n iterations can be presented as: {n X (N+6)}; where, N represents number of proteins in the network and n represents number of iterations. Based on this concept, the throughput of a PageRank algorithm for a large dataset using limited hardware resources can be calculated as shown in Figure 5(C).

A. *Methodology*

we developed our programmable hardware architecture using Verilog and simulated the design in Xilinx Vivado to verify the desired functionality of our design. Later, we followed a digital design flow based on CAD tools using a high-performance compact mobile computing plus (***CLN28HPC+***) process from TSMC ***28nm*** commercial PDK with 8 metal layers and supply voltage of ***0.9***V. We primarily focused on demonstrating programmable nature of our hardware to validate our novel computing concept and measured several design metrics to examine the hardware implementation benefits. Later, we used our computing strategy to handle complex operation of PageRank algorithm to analyze protein networks and observed throughput with increasing network complexity. We have maintained a uniform ***200*** MHz clock frequency in various stages of implementation like simulation, synthesis, and performance analysis. The simulation result of our proposed hardware architecture is depicted in Figure 4. In our testbench, we have considered six (***6***) messages (Left-1, TOP-1 to TOP-5) to be sent to our hardware unit. The current address of our hardware unit is ***5***, while the addresses of neighboring hardware units are ***2*** (Top), ***9*** (Bottom), Left (***4***), Right (***6***). Here, the opcode and the current destination of the left message-1 are "***Prog***" and "***5***" respectively; hence, the message will be decoded inside the SiteO rather than passing it to the right or bottom. On the other hand, the opcode and the present location of messages that are sent from the top side (TOP-1 to TOP-5) are "***Prog***" and "***9***". Therefore, these messages should be routed through the bottom port. The expected behavior in terms of programmability is achieved which is evident in simulation result shown in Figure 4. A single unit of our programmable architecture consumes only ***4.1*** mW power under TSMC ***28nm*** technology as reported in Table I.

Table I:
Design parameters of a single programmable core

Technology	TSMC 28nm
Process	HPC+
Metal Layer	1P8M
Voltage (VDD)	0.9V (nominal)
Package	Wire Bond
Area	6 mm2
Power	4.1 mW
Frequency	200 MHz
Gate Count	~98000

B. *Performance*

The latency of our matrix-vector multiplication process is appended in Figure 6 (A). Here, we have varied the number of rows (***N***) of the matrix of size (***N X M***) from 256 to 8192 and observed the respective latency. The results indicate that the time steps required to complete a matrix-vector multiplication is almost equal (***N+3 ≈ N***) to the number of rows in the matrix and it is independent of the number of columns in the matrix or the vector size. Figure 6 (B) demonstrates the throughput to analyze protein network utilizing PageRank algorithm. In this case, we have varied the number of proteins in the network from 1000 to 5000 and observed required times. For evaluation, we have conducted 100 iterations utilizing a ***200*** MHz clock frequency and leveraging only ***4096*** available hardware units. Our proposed computing methodology requires just ***213.6*** milliseconds to complete ***100*** iterations of PageRank algorithm to analyze a protein network comprising 5000 proteins.

IV. Conclusion

We presented a novel configurable hardware architecture designed for AI and data-intensive tasks. This architecture leverages a flexible interconnection scheme and a parallel organization of compute units, making it well-suited for large-scale, parallel, computationally intensive tasks. Additionally, we introduced the associated instruction set architecture (ISA) for programming and operation control. To demonstrate the efficacy of our hardware, we presented a case study involving a protein network search. Furthermore, we validated our design concepts through simulations conducted on a TSMC 28nm technology node. We also detailed our evaluation methodology and presented performance results for running the matrix-vector multiplication tasks of the PageRank algorithm within the context of protein network search. The combination of our innovative runtime-programmable architecture for compute-intensive tasks and its demonstrably fast performance suggests its significant potential for future applications.

V. References

[1] K. Akarvardar and H. . -S. P. Wong, "Technology Prospects for Data-Intensive Computing," in *Proceedings of the IEEE*, vol. 111, no. 1, pp. 92-112, Jan. 2023

[2] Z. Zhang, H. Wang, S. Han and W. Dally, "SpArch: Efficient Architecture for Sparse Matrix Multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, San Diego, CA, USA, pp. 261-274, 2020.

[3] N. Srivastava, H. Jin, J. Liu, D. Albonesi and Z. Zhang, "MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Athens, Greece, pp. 766-780, 2020.

[4] N. P. Jouppi *et al*., "In-datacenter performance analysis of a tensor processing unit," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, Toronto, ON, Canada, pp. 1-12, 2017

[5] B. Asgari, R. Hadidi and H. Kim, "MEISSA: Multiplying Matrices Efficiently in a Scalable Systolic Architecture," *2020 IEEE 38th International Conference on Computer Design (ICCD)*, Hartford, CT, USA, pp. 130-137, 2020

[6] Md Arif Iqbal, Srinivas Rahul Sapireddy, S. Dasari, Kazi Asifuzzaman, and M. Rahman, “A review of crosstalk polymorphic circuits and their scalability,” Memories - *Materials Devices Circuits and Systems*, vol. 7, pp. 100094–100094, Apr. 2024

[7] N. K. Macha, B. T. Repalle, M. A. Iqbal and M. Rahman, "Crosstalk-Computing-Based Gate-Level Reconfigurable Circuits," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 8, pp. 1073-1083, Aug. 2022

[8] P. Samant, Naveen Kumar Macha, and M. Rahman, “A Neoteric Approach for Logic with Embedded Memory Leveraging Crosstalk Computing,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 19, no. 1, pp. 1–16, Dec. 2022

[9] G. Alanis-Lobato, “Mining protein interactomes to improve their reliability and support the advancement of network medicine,” *Frontiers in Genetics*, vol. 6, Sep. 2015

[10] H.-W. Tang et al., “Next-generation large-scale binary protein interaction network for Drosophila melanogaster,” *Nature Communications*, vol. 14, no. 1, p. 2162, Apr. 2023

[11] D. F. Burke et al., “Towards a structurally resolved human protein interaction network,” *Nature Structural & Molecular Biology*, vol. 30, no. 2, pp. 216–225, Feb. 2023

[12] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine” *Computer networks and ISDN systems*, vol. 30, pp. 107-117, 1998.

[13] Lei, X., Liang, J. and Guo, L. ‘Identify protein complexes based on PageRank algorithm and architecture on dynamic PPI networks’, *Int. J. Data Mining and Bioinformatics*, Vol. 22, No. 4, pp.350–364, 2019