A Messaging based Intelligent Computing Approach for Machine Learning Applications

Mostafizur Rahman[+], Arif Iqbal[*], Srinivas Rahul[+]

University of Missouri Kansas City, Crosstalk LLC
Kansas City, MO, USA
rahmanmo@umsystem.edu

Abstract

Advances in machine learning (ML) algorithms and associated applications have fueled the inception of many new computing architectures with the aim to improve performance over traditional CPU and GPUs. However, the rapidly changing ML models and data variations landscape make it impossible for a purpose-built ASIC to keep pace. We propose a new computing platform, called messaging based Intelligent Processing Unit (mIPU), which can be programmed based on application needs without sacrificing performance. The vision is to provide FPGA like programmability and ASIC like performance for ML applications. At the center of the proposed architecture is a messaging-based scheme that allows interconnect programmability. Our evaluation results indicate over 200K images/sec throughput can be achieved for convolution operations. Our 28nm TSMC process-based evaluation of the hardware core indicates comparable power and density metrics against state-of-the-art GPUs.

(Keywords: Artificial Intelligence, Machine Learning Accelerator, Matrix Multiplication, Messaging Computing, Programmable Architecture)

1 Overview of the Architecture

The central idea of mIPU is to have a chip that can be reconfigured at run-time to behave as a custom-ASIC for each running AI applications to deliver the best performance. It revolves around a unique flexible virtual interconnection scheme where any computing core can be connected to another at run-time. An analogy of the proposed interconnect configurability is shown in Fig. 1(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 the 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 Fig. 1(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 themselves, called Sites hereafter, are very lite (capable of performing only essential operations like programming and arithmetic). A message originating from any of the Sites in a 16 Site configuration (4 rows and 4 columns- Fig. 1(right)), goes right or down depending on the destination. For example, a message originating from 1st row and 1st column (1,1) location destined for (2,2) goes downwards to (2,1) first, and then (2,1) passes it right to (2,2). Each Site, upon receiving or generating a message checks whether the destination is within the same row or not; if it is, then it sends the message to the right and to down otherwise. Eventually, through hopping Sites, a message reaches its destination. An example of matrix multiplication in m-IPU is shown in Fig. 4. The matrix operands are loaded parallelly from the top. All the destination locations are pre-set, so the messages for loading (11), (12), (13), etc., hop Sites, and reach their destinations. Once matrix A populates the Sites, then matrix B is loaded in cycles 5 onwards, and multiplications are performed. Afterward the resultant products are passed to other Sites for accumulation. Starting from cycle 6, results are generated and propagated. Both dense regular and sparse matrices can be handled with the same efficiency since the reconfiguration and computation scheme is not rigid for certain data type/shape/size.

The SiteOs are responsible for both computation and message passing. When a message arrives at SiteO, it first checks whether the destination of the message is its address, and if it matches, then the message is decoded and the instruction embedded within the message is executed, otherwise, the message is passed on. To load a 2 x 2 matrix in 4 SiteOs, 4 messages need to be sent to those 4 specific SiteOs. The SiteOs are capable of basic arithmetic (e.g., add, mult, sub) operations. They are aware of their neighbors (i.e., addresses of neighbor SiteOs in right, left, up, and down (Fig. 1 (right)) are stored in each SiteO). SiteOs also store a value and destination address to generate messages. If the messages are to be routed/passed downward, those messages are labeled as *Tile message* and if they are passed rightward (within the same SiteO row), those are labeled as *Local messages*. The phase when stationary values are first loaded is called programming. To distinguish between programming and operation, the opcode values act as guides. The message encoding scheme is shown in Fig. 3B. For our example in Fig. 4, the SiteO located in (0,0) position in the 16 Site based Tile organization (Fig. 1(right)) receives a message whose opcode is *PROGDS,* 11 as value, *ACCUMS* as next opcode, and row 0, column 3 as the next destination, which means this SiteO should store 11 as its stationary value, enable *down* stream flag to stream operands downwards and also store *ACCUMS* in the opcode field and 03 in the destination field for future messages originating from this SiteO. Streaming and message forwarding are two different tasks; in case of streaming, the SiteO receiving the message send it to its preferred neighbor by updating the message, whereas in forwarding, the SiteO just behaves as a buffer to pass messages without intervention.

The m-IPU engine only needs to be interfaced with the memory to input instructions and output data through the memory to the outside world (Fig. 3A). A host CPU is required (similar to GPUs and other accelerators) to interpret high-level language (e.g., C, Python, etc.) and translate them into messages that m-IPU can operate upon (inside the m-IPU all communication between computing and storage elements is through messages). The memory is an L1 cache segmented into message storage and output data sections. The control unit ensures that the messages and data are synchronized. Inside the m-IPU engine, there is an array of Blocks. The Blocks communicate with each other through local and global buses. A Quad is a collection of 4 Blocks, and a Block is a collection of 16 Tiles (Fig. 7B, C). Each Tile consists of 16 Sites (Fig. 3C). Sites are of two types: SiteO and SiteM. A computing core to perform arithmetic and logic operations is called SiteO in m-IPU. The SiteOs are the core elements and are analogous to Threads of GPUs or the Processing Elements (PEs) of TPUs. The hierarchy of Quads, Blocks, Tiles, and Sites allows task distribution and parallel computing. Associated with each SiteO is a small 8 word memory buffer to store next set of instructions. The SiteOs also contain SRAMs to store weights. Each block also contains distributed embedded memory elements to store further instructions. For ResNet-50, in total, 33MB of embedded memory is required with 128 memory banks. The minimum bandwidth required for 1Ghz frequency-based operation would be 128Gbps. For our simulated prototype, the frequency was kept at 300MHz at 28nm node.

2 Evaluation Methods & Results

We have implemented a miniaturized prototype of in post layout simulations and also implemented an interpreter to convert intermediate representation of higher-level models to machine codes. Our hardware implementation was based on the architecture outlined earlier. Implemented SiteOs were capable of matrix multiplication, accumulation, and comparison operations. We used TSMC 28nm for characterization. We used Cadence Genus for synthesis. The FIFO size was 64 bits x 8 words, and the ALU was capable of handling 32bit floating point arithmetic. The simulation results showing proper functionality is shown in Fig. 12 where a SiteO is programmed first and then is used for multiplication and streaming values. Messages were incoming only from left. Our evaluation results are shown in Fig. 5. For throughput calculations, the total number of Sites were assumed to be 16384 and the frequency to be 1GHz. In software, we implemented an interpreter and a performance emulator that can generate machine code from lower level IR produced by Glow compiler and at the some time predict performance. For performance estimation, the total number of cycles required were calculated iteratively based on each running program like convolution, softmax, maxpool and others. Fig. 5 shows results from 1st convolution of ResNet-50 for 228x228x3 images with each pixel having 32bit floating point values. As it can be observed, the throughput increases with higher batch size. For a batch size of 8, over 600K images can be processed per second. Our initial results also indicate high throughput for ResNet-50; over 70K images per second can be processed for batch size 1.

Due to the unique configurability approach, distribution of memory and computing resource utilization is maximized, and when compared with state-of-the-art, huge benefits are expected for Deep Neural Networks. For software, the requirements are resource allocation and message generation, which can be done by a CPU at run-time.

References

[1] N. K. Macha, et. Al., Crosstalk Noise based Configurable Computing: A New Paradigm for Digital Electronics, IEEE international system-on-chip conference,2021.

[2] N. K. Macha, et. Al, Fine-Grained Polymorphic Circuit Framework in Crosstalk Computing, IEEE Transactions on VLSI,2022 (pending publication).

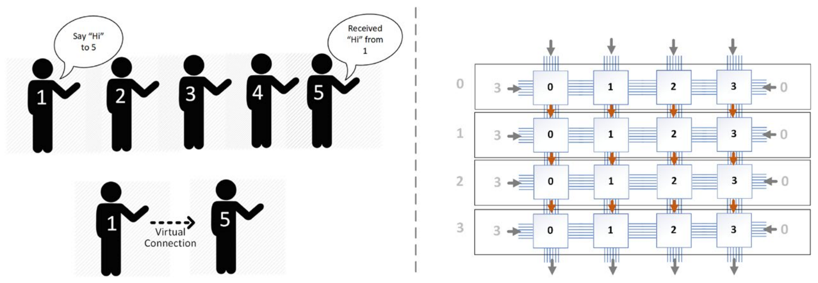
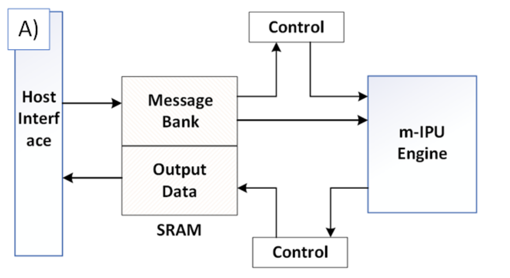
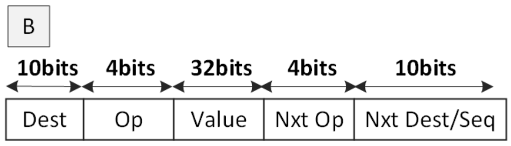


Figure 1. Left shows an analogy of virtual connection through human chain. Similar concept is extended to connect computing cores (Sites) on the right. There are 4 SiteOs in each row that are connected from left to right with the rightmost node connecting the leftmost one from right. The SiteOs in each row are connected vertically as well in columns. This configuration allows any of the 16 SiteOs to communicate with another.

Process	TSMC 28nm HPC+
Frequency	300 MHz (Single Clock)
Area	8035 um2 (65% density)
Power	1.2 mW
Timing	8ps (+ slack)
VDD	0.9 V
Metal Layer	1P8M

Figure 2. Post layout synthesis results for SiteO.





Dest: Destination Address
Op: Op Code to perform an operation
Value: The value to be stored/operated upon
Nxt Op: Op code to be embedded in the message that will be generated from this SiteO
Nxt Dest/Seq: Destination address or Sequence # to be embedded in the message that will be generated from this SiteO

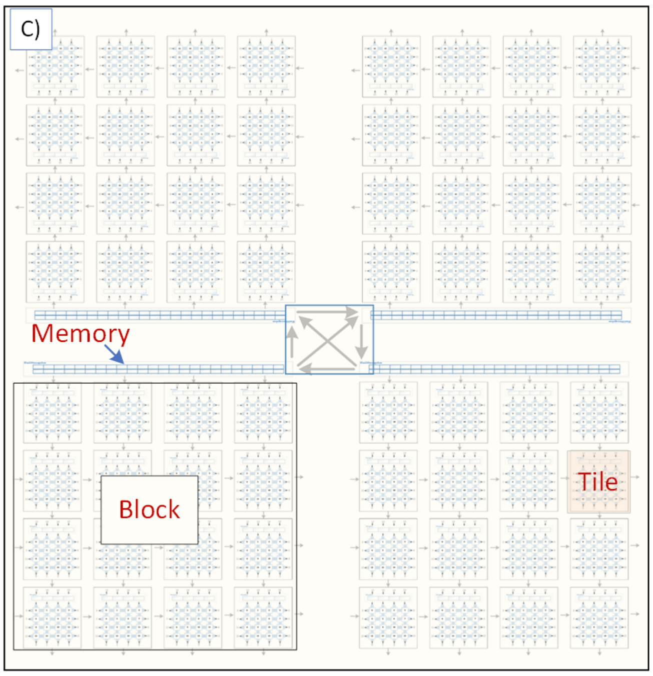


Figure 3. A) Chip overview showing the key components. B) diagram shows encoding of messages. B) Snapshot of the m-IPU engine showing 4 Blocks connected through a bus. A larger chip will have many Blocks, each Block has dedicated embedded memory to store instructions and data.

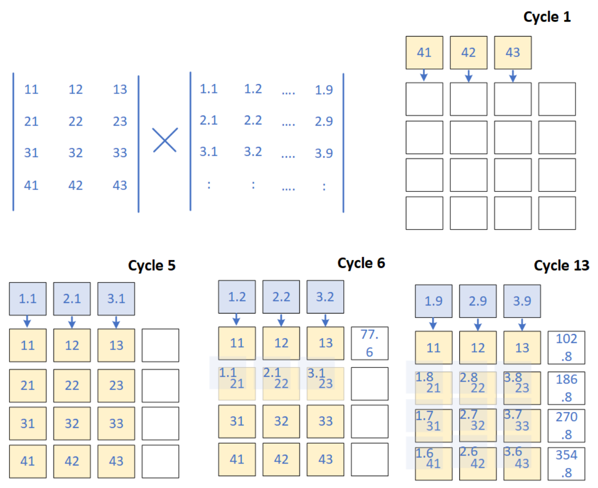


Figure 4. Matrix multiplication example in m-IPU. Each row has a common bus to share results.

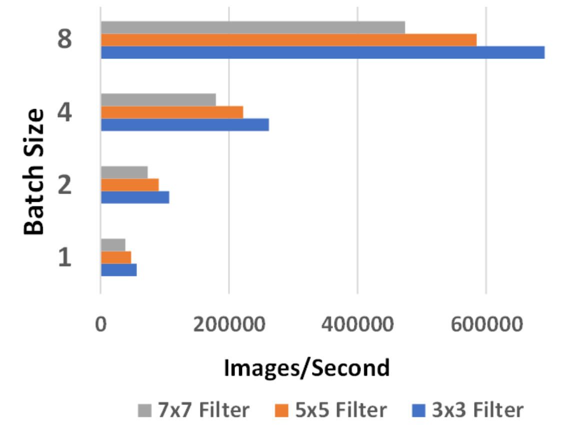


Figure 5. Throughput vs batch size. Each image is 228x228x3. 32 bit FP data is considered. A stride of 2 and pad of 2 was considered. Filter count was 64. With higher batch size better throughput is observed.