Optimizing Artificial Intelligence Performance with Morphable Parallel Computing Architectures

Introduction and Motivation

- Artificial Intelligence (AI), the field associated with modeling human intelligence for computation, has skyrocketed over the past few years
- AI has an unmatched **ability to make deep connections** between arbitrary inputs and outputs, often in ways researchers can't understand
- The most popular AI Structure is the **Neural Network (NN)**

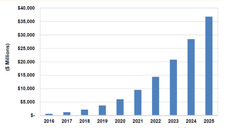


Figure 1: AI Revenue Growth Pattern[1]

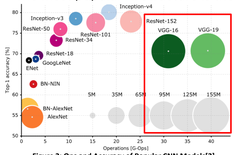


Figure 2: Ops and Accuracy of Popular CNN Models[2]

- As seen in Figure 2, Neural Networks, such as VGG and ResNet, are extremely accurate, but because of their enormous size, require **enormous computational resources**
- Each of these require **greater than 25x10^9 operations** per forward pass of data
- To develop increasingly complex AI models, **faster hardware architectures** must be developed to handle the unprecedented demand for floating point operations per second (flops)

State of AI Hardware

The propagation of inputs through the weights of a Neural Networks is computed through **matrix multiplication**.

Here lies the computational overhead of Neural Networks, because operations (multiplications and additions) per second increases linearly with added nodes and added layers, making this a quadratic problem.

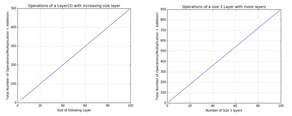


Figure 3: Operation Cost per Layer and Nodes [10]

- Therefore, AI Hardware research is focused on **optimizing large-scale matrix multiplication**
- The most popular processors are graphics processing units (GPU)'s
- However, recently there have been several innovations in the realm with **Google's Tensor Processing Unit** (TPU) and **Intel's Nervana** Chip

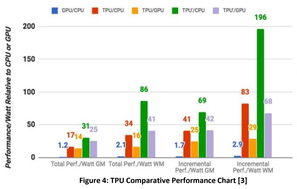


Figure 4: TPU Comparative Performance Chart [3]

- Nervana requires binary weights rather than 8, 16, or 32 bit numbers, allowing for bundles of XOR gates to act as a powerful matrix multiplier, but this comes at the cost of added precision
- The TPU retains floating point precision while maintaining speed through its **Systolic Array**-Based Processor
- Compared to CPU's and GPU's, the **TPU** vastly **outperforms** in terms of **flops** and **flops/watt**
- The **GPU's power consumption** is a large **hindrance** for **server providers** and mobile **AI robotics**

Systolic Array Background

- A Systolic Array is a **highly parallelized** computer architecture that uses a number of processing elements (PE) which multiply and accumulate (MAC) the dot products of the end matrix
- In the case of the TPU, the PE's were arranged in **rectangular formation**
- Consider the example, **C = AxB**, where A and B are 3x3 matrices to be multiplied

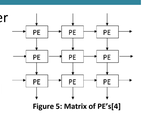


Figure 5: Matrix of PE's[4]

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & a_{33} \end{bmatrix} \qquad A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

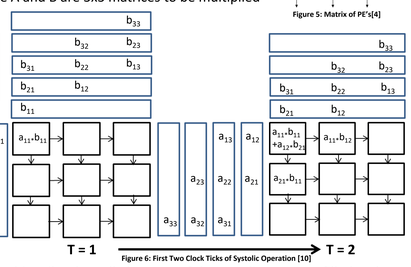


Figure 6: First Two Clock Ticks of Systolic Operation [10]

- This process continues until **T = 7**, or arbitrarily when all elements of the systolic array are filled and the input data has been emptied.
- Compared to a normal CPU, which would require **T=27** to multiply the matrices, this system's concurrency drastically improves total latency

Scalability Issues and Focus

While the systolic array is state of the art, there are several perform hindrances at scale:

1. **Repetition** – For matrices which exceed the size of a systolic array's row and column size, the systolic array is forced to **repetitively cycle through the larger matrices** and perform block matrix multiplication. On^2 problem
2. **Memory** – To simply enlarge the systolic array for larger matrices is a faulted strategy because memory loadup latency **quadratically increases** with the side length of the matrix
3. **Clock Cycles** – The **time required** for a systolic array to finish computation is **dependent on the size of the systolic array** itself, so the larger the systolic, the longer it takes to complete

***Developing a sizing algorithm which minimizes these three functions is the central focus of this research**

- **It is imperative that the algorithm not create additional latencies, but minimizes the 3 stated hindrances**
- **The Systolic Array must be of 1-D Size to handle matrix-vector multiplications**

- Neural Network Training often contains **thousands of large-scale matrix** multiplications, since training sets often contain >1000 examples and >10 epochs
- This is why Cloud Computing is a rising industry, but **power costs** for server providers and **hourly compute costs** for consumers is an issue
- The development of a faster systolic array, which has been proven to be more power-efficient than GPUs, could help both server providers and consumers by lessening server-time usage.


Figure 7: Cloud Computing[5]

System Modeling

Latency Modeling

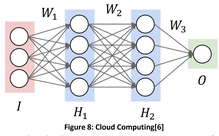


Figure 8: Cloud Computing[6]

Between each of layer in a Neural N, there is a multiplication between a **weight matrix** and propagated **input vector**.

$$H_{1_{4x1}} = W_{1_{4x3}} * I_{3x1}$$
$$H_{2_{4x1}} = W_{2_{4x4}} * H_{1_{4x1}}$$
$$O_{1x1} = W_{3_{1x4}} * H_{2_{4x1}}$$

When using a systolic array for NN multiplication, holistic run-time is a sum of each layer's latency. The goal of modeling is to **minimize holistic latency** by **reshaping the systolic array's side lengths**.

In the below equations, **m = systolic array row size, A = weight matrix, B = input vector**

Repetition

When the weight matrix is larger than the dimensions of the systolic array, it has to be cut into smaller portions.

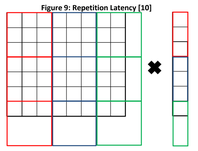


Figure 9: Repetition Latency [10]

$$Repetitions(m) = ceiling\left(\frac{A_{row}}{m}\right) * ceiling\left(\frac{A_{col}}{m}\right)$$

Clocks

For the multiplication of a matrix and vector on a systolic array, completion time, or clock cycles is a function of size.

Propagation of data through the systolic array is a function of its row size, and as the number of elements increases, so does run-time.

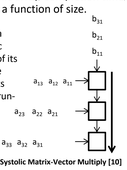


Figure 10: Systolic Matrix-Vector Multiply [10]

$$Clocks(m) = 2 * m - 1$$

Memory

Between each cycle, time is required to load the matrices and vectors from memory.

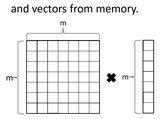


Figure 11: Matrix Multiplication [10]

The input matrix is 2-D, so memory grows quadratically with size.

$$Memory(m) = m^2 + m$$

Latency for Layer-wise Multiplication can be approximated by:

$$LayerLatency(m) = Repetitions(m) * (\,Memory(m) + Clocks(m)\,)$$

The goal is now to find to **find the size** (m), which will **minimize this function**. Rather than take a derivative and find x-axis intersections, a **1-D search algorithm** finds the smallest latency and its corresponding systolic size.

In the side case, the optimal size for multiplying a 512x512 matrix and a 512x1 vector is 170 Processing Elements.

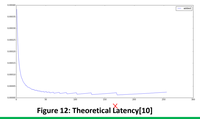
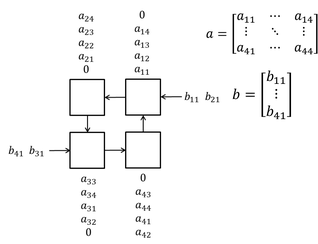


Figure 12: Theoretical Latency[10]

Circular-Flow Systolic



$$a = \begin{bmatrix} a_{11} & \cdots & a_{14} \\ \vdots & \ddots & \vdots \\ a_{41} & \cdots & a_{44} \end{bmatrix} \qquad b = \begin{bmatrix} b_{11} \\ \vdots \\ b_{41} \end{bmatrix}$$

Figure 13: Circular Multiplication of a and b [10]

- One of the largest problems with the systolic array is that clock **ticks grows as a function of twice its side length**.
- Most of this time is spent moving the vector's elements across the systolic PE rather than dot product accumulation
- A **novel**, circular flow linear systolic is proposed, whereby data is **injected at both the beginning and midpoint** of the systolic array
- Additionally, instead of leaving the systolic array, data is forwarded back to the first element
- This **reduces clock ticks while keeping other factors like repetition and memory constant**

The minimum number of clock cycles required to multiply the two matrices is:

$$Clocks(m) = 1.5m\ -1$$

Theoretically, in the case of a 256x256 and 256x1 matrix multiply, the repetition speedup with a circular-flow systolic is around 1.333x

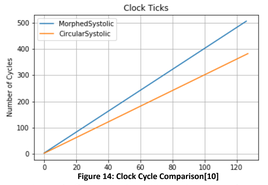


Figure 14: Clock Cycle Comparison[10]

Similar to the Morphable Systolic System, the optimum shape is first calculated based on the sizes of matrices in the instantaneous layer.

However, holistically, in order to find the optimum size for the entire network, the network optimum size is a weighted average of individual layer latency and size.

$$OptimumSysSize(m) = \frac{\sum_{input\ layer}^{output\ layer} OptimumLayerSize\ * LayerLatency(m)}{\sum LayerLatency(m)}$$

Parallelization

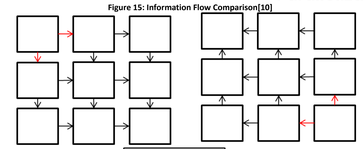


Figure 15: Information Flow Comparison[10]

- It is conventionally assumed that data is *pushed from one PE to its neighbors*

In Python though, a systolic array matrix is created and *rather than push inputs* from top left to bottom right, *inputs are pulled* from bottom right to top left

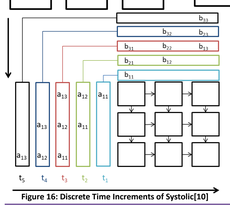


Figure 16: Discrete Time Increments of Systolic[10]

In Figure 16, the information within the systolic matrix can be modeled by a set of discrete time steps. A while loop cycles through these steps until multiplication is finished.

To reduce PE complexity, each PE evaluates 4 rules:

1. If(NorthValue and WestValue aren't None): **perform a MAC operation**
2. If(NorthValue is None, but NorthNeighbor's NorthValue isn't None): **NorthValue = NorthNeighbor's NorthValue**
3. If(WestValue is None, but WestNeighbor's WestValue isn't None): **WestValue = WestNeighbor's WestValue**
4. If(#MAC's for all PE's = col_size of A): **stop systolic array**

Design Challenges and Goals

- In order to test the efficacy of the proposed algorithm and Circular-Flow Systolic Structure, a high-level simulator was sought to be designed.
- Previous works have attempted such a task [5], but not with the programmability incurred with a higher-level programming language like Python, which is the suggest language.

Challenges	Requirements
Parallelization	Systolic Array's are effective because of their parallel nature, and while all higher-level languages are executed asynchronously and line-by-line, the proposed system will have to mask any iterative nature incurred by trying to model the systolic array.
Memory Access	In ASIC and FPGA applications, it takes time to lead floating-point numbers from memory. In fact, it was stated that compared to a 512x512 size, the 256x256 systolic array performed much faster, simply because less memory had to be retrieved per clock cycle [8].
Processing Element Complexity	Processing Elements have two functions: Multiply and Accumulate or Pass on Inputs to Neighbors. Therefore, to model simplicity, the Processing Element Objects must be written concisely.

Data Injection

- To achieve the desired data-stagnation procedure as seen in conventional systolic arrays, two Python operations were constructed.
- In either case, **sys** is a matrix of PE's, and **it** is a global cycle counter, and **i** is a counter while iterating through the side length of the array
- For a Normal Systolic Array:

sys[i][0] = A[it – i][i]
sys[0][i] = B[it – i][i]

- For a Circular-Flow Systolic Array:

sys[i][0] = A[it – i%len(sys)/2][i]
sys[0][i] = B[i][it – i%len(sys)/2]

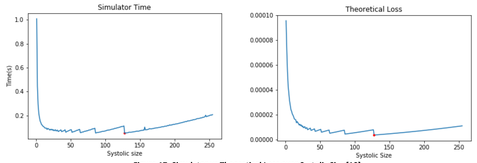


Figure 17: Simulator vs Theoretical Loss over Systolic Size [10]

- In the above case, where a 256x256 matrix and a 256x1 vector are being multiplied, the **simulator's elapsed time and optimum size matches that of the equations almost perfectly.**
- The only difference is the greater increase in latency per unit increase in systolic size after a size of 120, but this has no bearing on the recognition of the optimal size.

```
Normal Systolic
2048x2048 by 2048x1
Adjusted time difference: 15.482391119003296
Pseudotime difference: 15.483251571655273
Cutout from For Loop: 11.92989945411682l
Clock cycles: 4095
Pseudototal: 15.479477834643316
-----------------------
Circular Systolic
2048x2048 by 2048x1
Adjusted time difference: 10.60096484756469?
Pseudotime difference: 10.610525131225586
Cutout from For Loop: 6.452003717422485
Clock Cycles: 3071
Pseudototal: 10.607863902036392
-----------------------
Speedup: 1.9632897654x
```

Figure 18: Statistical Comparison of Systolic's[10]

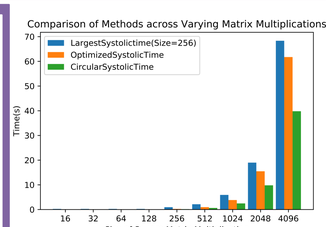
- A **critical component** of the simulator's execution accuracy is its **subtraction of any repetition**
- When injecting data or moving data within the systolic array, the simulator subtracts the time required to loop through elements
- This is because a parallel system doesn't require a loop; **the time for one PE to operate** is **equal to that of the entire array**
- Figure 18 shows the CircularSystolic's true speed enhancement compared to a Normal Systolic.

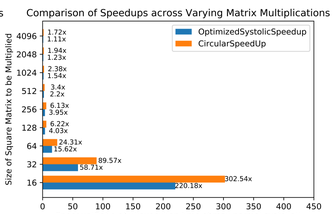
Simulator Design and Testing

Each of the simulations were run on Amazon EC2 Servers.

In the last test, the MLP portions of the most popular Convolutional Neural Networks, **VGG-19**, **ResNet**, and **AlexNet** were tested.

Architecture & **Matrix** Sizes[11]:
Input: 512x1 size
Layer1: 4096 nodes (4096x512)
Layer2: 4096 nodes (4096x4096)
Layer3: 4096 nodes (4096x4096)
Layer4: 1000 nodes (1000x4096)
Layer5: 300 nodes (300x4096)





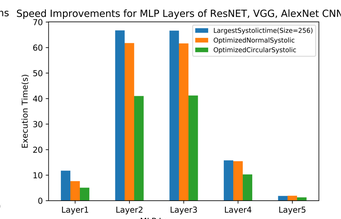


Figure 19: Systolic Performance Analysis [10]

Data Analysis

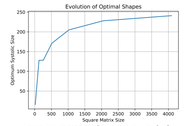


Figure 20: Approach of Optimal to Max [10]

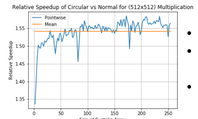


Figure 21: Relative Speedup of Circular [10]

- As seen in Figure 20, simply **optimizing the size** of a Systolic Array with the proposed algorithm leads to **speedups of 1.54x** when multiplying matrices and vector's which **quadruple the size of the array**
- However, when approaching the largest of matrices, the algorithm's benefits begins to decrease, since at **a 4096x4096 matrix multiply**, the speedup is only around **1.11x**
- This is because when specifying a max # of PE's, which is necessary since **hardware resources are finite**, memory approaches a threshold, while repetitions and clock cycles continue to increase
- As seen in Fig. 20, the **optimum size quickly approaches the max # of PE's**
- However, the Circular Systolic circumvents this issue with a **constantly smaller number of clock-ticks compared to a Normal Systolic Array**
- As seen in Figure 21, for a 512x512 multiplication, regardless of spatial dimension, the **speedup** for a Circular Systolic **is constant** compared to a Normal Systolic
- The **true test** came with Figure 19, and the comparison of the various Systolic Structures versus the Multi-Layer-Perception (MLP) Layers of the largest, most popular Convolutional Neural Networks
- Each CNN model uses **the same MLP architecture**, and across each layer, **both** the **Optimized Systolic** and **Circular Systolic** showed **speedups** over **the traditional, maximum size Systolic**
- This was a unique test of the Systolic's Performance since unlike the tests, the matrices weren't always square, but rectangular.
- The algorithm accounted for this and produced optimal sizes dissimilar to that of square matrices.

Summary and Conclusion

- **Artificial Intelligence** has been **growing at an exponential rate** over the past few years, which has warranted the research of faster, more computationally-capable hardware
- AI is reliant upon **fast matrix multiplication**, which is where the **GPU** has excelled, but research at **Google** has shown that the **Systolic Array is a promising option** for its **flops** and **flops/watt performance**
- The Systolic Array is highly parallelized, but when bound in the spatial dimension, can significantly suffer with large-scale matrix multiplication
- The intent of the research was to **develop a cost function for any systolic array**, based on a given matrix-vector operation, and **output the optimum systolic array shape**
- A mathematical model was created and an algorithm in Python performs a 1-D search to find the optimal size
- Additionally, a unique Systolic Structure called the **Circular Flow Systolic** is proposed to **minimize** the Systolic's **clock cycles** while keeping other factors constant
- Due to hardware limitations, **a simulator was designed** in Python to evaluate the proposed algorithm and Circular Systolic Structure
- It was shown that **across various square matrix multiplications** and rectangular multiplications as found in popular CNN Models, both the **algorithm** and the **Circular Systolic provided significant speed enhancements**
- **Applications of this technology** might find itself in **mobile robotics**, where matrix multiplication latency is required to be minimized or in **Cloud Computing**
- Companies such as Amazon and Google provide hardware for developers to train their AI models, and with faster, **more electricity efficient systems**, the **providers would spend less money** hosting the servers, and **consumers would pay less** for hourly consumption as training time would be lessened

Preliminary FPGA Development

- While the focus of this project was to develop an algorithm for enhancing matrix-vector multiplication, a true test of its efficacy is to test on a **Field Programmable Gate Array (FPGA)**
- FPGA's are programmable chips that are commonly used for prototyping, but in this scenario, its **morphability** makes it an ideal candidate for the proposed algorithm
- FPGA's are also significantly **more power efficient than GPU's**, but lower-end FPGA's drastically suffer with memory access


Figure 22: Mojo FPGA [7]

The **Mojo FPGA** is a simplistic starting point, equipped with a **Spartan VI** chip and several header pins for IO and additional memory installation

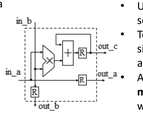


Figure 23: PE at FPGA Scale [8]

- Unlike in Python, where commands are executed sequentially, FPGA's execute lines instantaneously
- To the Left, is an example of a PE on an FPGA, simply consisting of an adder, multiplier, RAM, and registers for the outputs
- A simplistic **example with multiplying 2, 3x3 matrices** was conducted, and the execution time was **46ns** at a clock speed of **128 MHz**
- A CPU executing **C++ multiplication**, chosen for the languages speed, takes **0.054s**.

Current/Future Research

While the systolic array's were tested on the MLP portions of the CNN's, the preceding **convolutional layers** weren't accounted for.

Convolution is the **process of element-wise multiplying** a section of a matrix, in practical cases, **images**, with a **kernel**, and **summing the result**

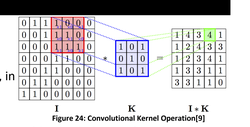


Figure 24: Convolutional Kernel Operation[9]

- While seemingly unrelated to matrix multiplication, convolutional layers are **nothing both dot products** between **unraveled images** and a **weight-matrix like in an MLP Layer**
- The largest difference is that with convolution, **matrices** are multiplied **rather than a matrix and a vector**, so the current architecture must be **slightly changed to account for the added dimension**
- Additionally, research into **FIFO Buffers** and memory speedups is a necessary task, since hardware resources have **limited processing elements** and therefore are stuck with **constant memory latency while approaching the max size**
- **FPGA implementation** of the algorithm **must be finished** to show tangible enhancements over current hardware

Bibliography

Resources(Abridged)
1. H. T. KUNG, C. E. LEISERSON, Systolic arrays for VLSI, Introduction to VLSI systems, (C.Mead, L.Conway, eds.), Addison-Wesley Ltd., Reading, MA, 1980.
2. H. T. KUNG, Why systolic architectures?, Computer, 15 (1982), 37-46.
3. M. CHEN, A design methodology for synthesizing parallel algorithms and architectures, J Parallel Distributed Comput., (1986), 461-491.
4. A. L. DECEGAMA, Parallel processing architectures and VLSI hardware, Prentice Hall, Newtersey, Vol. 1, 1989.
5. J. A. FORTES, R. S. FU, B. W. WAH, Systematic design approaches for algorithmicallyspecified systolic arrays, (V. Milutinovic, ed.), Computer Architectures, Elsevier Ltd., New York, 1988, 454-494.
6. H. V. JAGADISH, T. KAILATH, A family of new efficient arrays for matrix multiplication,IEEE Trans. Comput., 38 (1) (1989), 149-155.
7. C. LANGAUER, A view of systolic design, In: Proc. International Conference Parallel ComputingTechnologies, (N. N. Mirenkov, ed.), Novosibirsk'91, World Scientific, Singapore, 1991,32-46.
8. G. -J. LI, B. W. WAH, The design of optimal systolic arrays, IEEE Trans. Comput., 34 (1)(1985), 66-77.
9. Schmidt, R. O., R. Roy, and T. Kailath. "Introduction to CIM-OSA." Open System Architecture for CIM (1989): 39-41. Gird Systems. 7 July 1989. Web. 28 Feb. 2016.
10. Gibson, Chris Nicholson Adam. "A Beginner's Guide to Eigenvectors, PCA, Covariance and Entropy." *Eigenvectors, PCA, Covariance and Entropy - Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, Skymind*
11. Siddarth Das.CNNs Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet andmore ...Nov. 2017.

Figure Credits
1. Michael Feldman. "Market for Artificial Intelligence Projected to Hit 36 Billion by 2025". In: 3 (2016), p. 2.
2. Michael Feldman. "Market for Artificial Intelligence Projected to Hit 36 Billion by 2025". In: 3 (2016), p. 2.
3. Extreme Technology. Nvidia claims Pascal GPUs would challenge Google's Tensor-Flow TPU in updated benchmarks. Apr. 2017.
4. North California Southern University.CSC/ECE 506 Spring 2013/1a sp. Debruary2013.
5. IBM.What is Cloud Computing. June2017.
6. Ashish Sukhadeve.Understanding Neural Network: A beginner's guide. Aug. 2017
7. Embedded Micro.Embedded Micro's MOJO V3 Chip. Aug. 2017.URL:https://embeddedmicro.com/products/mojo-v3
8. Aahan Peris. Digital Design of systolic array architecture for matrix multiplication Dec. 2017.
9. Andrej Karpathy.Convolutional Neural Networks (CNNs / ConvNets). Aug. 2015.
10. Figure Taken/Created by Student