Design and Simulation of a TPU-Style 8×8 Systolic Array for Matrix Multiplication

Abstract

This project involves the design and simulation of a specialized matrix multiplication accelerator modeled after Google's Tensor Processing Unit (TPU). An 8×8 systolic array architecture is implemented in Verilog to perform matrix multiplication of two 8×8 matrices. Each Processing Element (PE) in the array executes signed multiply-and-accumulate operations in a pipelined fashion, mimicking the behavior of a TPU's matrix unit. In our approach, the elements of matrix **A** (the "weight" matrix) are fed into the array horizontally from the left, while the elements of matrix **B** (the "data" matrix) are introduced vertically from the top. The goal is to demonstrate how a systolic array can accelerate deep learning computations by performing many operations in parallel. A testbench supplies two 8×8 matrices to the systolic array and verifies that the output matrix matches the expected product. The report covers the need for neural network accelerators, an overview of systolic arrays for matrix math, detailed descriptions of the Verilog modules (PE, 8×8 array, top-level, and testbench), block diagrams of the architecture, the simulation setup and results, and conclusions with potential enhancements. The successful simulation confirms the design's correctness and illustrates the efficiency of TPU-style hardware for matrix operations.

Introduction

Deep learning tasks involve intensive linear algebra computations, especially matrix multiplications for operations like multiplying weight matrices by activation vectors in neural networks. General-purpose processors (CPUs) and even GPUs often face bottlenecks when executing these workloads, due to limited parallelism and memory bandwidth constraints. Specialized hardware accelerators have emerged to meet this need by providing massive parallel multiply-accumulate capability with efficient dataflow. Google's Tensor Processing Unit (TPU) is a prime example of an accelerator designed specifically for neural network computations. A TPU is essentially a matrix processor that contains a large array of arithmetic units connected in a grid (a systolic array) to perform matrix multiplication with high throughput (TPU architecture | Google Cloud). For instance, the first-generation TPU contains 65,536 (256×256) 8-bit MAC units operating in unison (An in-depth look at Google's first Tensor Processing Unit (TPU) | Google Cloud Blog), allowing it to multiply two 256×256 matrices in a single cycle (with appropriate data setup).

The motivation for this project is to study and replicate the core concept of a TPU's matrix multiply unit on a smaller scale. By implementing an 8×8 systolic array in Verilog, we can explore how data flows through a network of PEs to compute a matrix product efficiently. In our design, matrix **A** is treated as the "weight" matrix and is fed horizontally from the left, while matrix **B** is considered the "data" matrix and is fed vertically from the top. This custom hardware accelerator loads matrix data, pumps it through the array in a wavefront manner, and produces the result with minimal reliance on off-chip memory during computation. Such a design avoids the Von Neumann bottleneck – by **reusing data within the array** (each element is fetched once and then flows through multiple PEs) (Systolic Array Data Flows for Efficient Matrix Multiplication in Deep Neural Networks). The following sections describe the systolic array architecture and its implementation details, followed by the simulation methodology and results that validate the design.

Systolic Arrays for Matrix Multiplication in Neural Networks

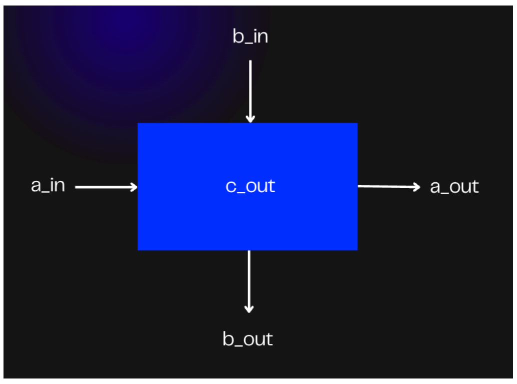
Systolic arrays are a class of parallel architectures consisting of a grid of processing elements that rhythmically compute and pass data to each other (analogous to the pulsing of a heart, hence "systolic") (Systolic array - Wikipedia). Each PE in the array performs a simple operation (such as multiply-add) and passes its computed result to neighboring PEs in lockstep with a global clock. This structure is particularly suited for matrix multiplication, which requires a large number of repeated multiply-accumulate operations. In a systolic array configured for multiplying matrices, the two input matrices are fed into the array from two perpendicular directions. In our updated design, **matrix A is fed in horizontally from the left**, and **matrix B is fed in vertically from the top**. As these values propagate through the array, each PE multiplies a pair of elements (one from each matrix) and accumulates the partial sum for one output element. By the time the data wave has swept through the array, every PE will have computed one element of the output matrix.

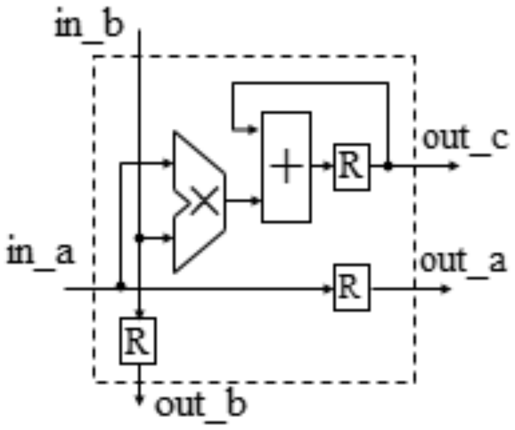
One major advantage of systolic arrays is data reuse: a single data element is used in multiple operations as it moves through neighboring PEs, reducing the need to fetch it repeatedly from memory (TPU architecture | Google Cloud). This is a key reason why TPUs and similar accelerators achieve high efficiency – once the matrix data is loaded into the array, the multiplications and accumulations happen on-chip with no further memory access until the result is produced (TPU architecture | Google Cloud). For neural network workloads, this means that the systolic array can compute an entire layer's matrix operations with minimal I/O, greatly accelerating performance. In our 8×8 systolic array – a miniature version of a TPU's matrix unit – the same principles apply: as matrix **A** flows from the left and matrix **B** descends from the top through the grid of PEs, the architecture achieves 64 parallel MAC operations every clock cycle (in steady state), offering significant efficiency for matrix-matrix multiplication.

Design Architecture and Verilog Module Descriptions

To implement the matrix multiply accelerator, the design is divided into modular Verilog components. The hierarchy includes: (1) the **Processing Element (PE)**, which is the basic compute node; (2) the **SystolicArray8x8**, which instantiates an 8×8 mesh of PEs and handles data movement between them; (3) the **MatMul8x8_Top** top-level module, which feeds input matrices into the array and collects the results; and (4) the **tb_MatMul8x8** testbench, which provides stimulus (the matrices to multiply) and verifies the output. Block diagrams are used to illustrate the PE design and the overall 8×8 array structure for clarity.

Processing Element (PE)





The PE is the fundamental building block of the systolic array, responsible for performing the multiply-accumulate operation. Each PE has two input data ports and two output forwarding ports. In our design, these are defined as:

- **Input A:** the data value coming from the left (corresponding to an element of matrix A).
- **Input B:** the data value coming from above (corresponding to an element of matrix B).
- **Output A:** the forwarded A value sent to the PE on its right, so that the same matrix A element can be used by subsequent PEs in that row.
- **Output B:** the forwarded B value passed to the PE below, ensuring that the same matrix B element is available to the subsequent PEs in the column.
- **Output C:** the partial sum result accumulated by the PE. As the systolic operation progresses, each PE's accumulator eventually holds the final computed value of one element of the output matrix C.

Internally, the PE completes four actions each clock cycle (after initialization): **(1) Fetch** the current inputs (A from the left and B from above); **(2) Multiply** the inputs; **(3) Add** the product to the running sum stored in an accumulator register; and **(4) Forward** the A value to the right and the B value downward. The accumulator is initially set to zero and is updated with each new pair of inputs. For instance, using 8-bit two's complement arithmetic, the multiplier generates a 16-bit product and the accumulator is sized to prevent overflow. This pipelined operation ensures that by the end of the computation, every PE has correctly computed the sum of products for its designated output element.

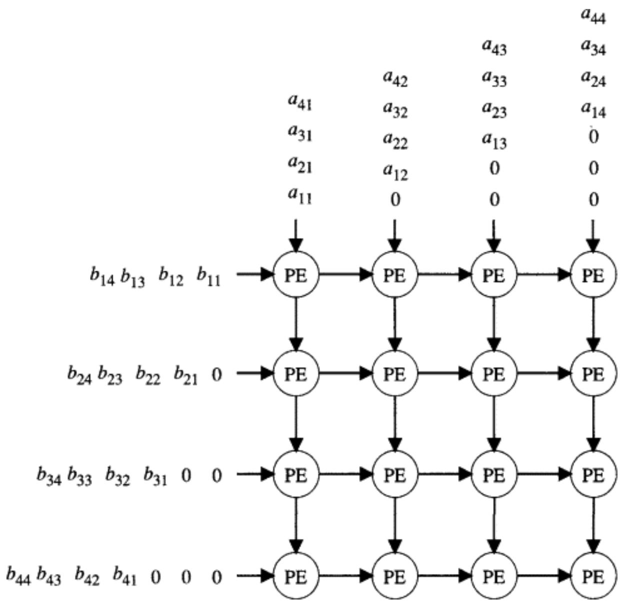
8×8 Systolic Array (SystolicArray8x8)

Figure: Block diagram of the 8×8 systolic array architecture. The array comprises 64 interconnected PEs. ***Matrix A*** *elements are introduced from the left via a* ***Weight*** *input queue (flowing row-wise), and* ***Matrix B*** *elements are injected from the top through a* ***Data*** *input queue (flowing column-wise). Each PE performs a multiply-accumulate operation, and the partial sums are eventually assembled as the output matrix C at the bottom-right of the array.*

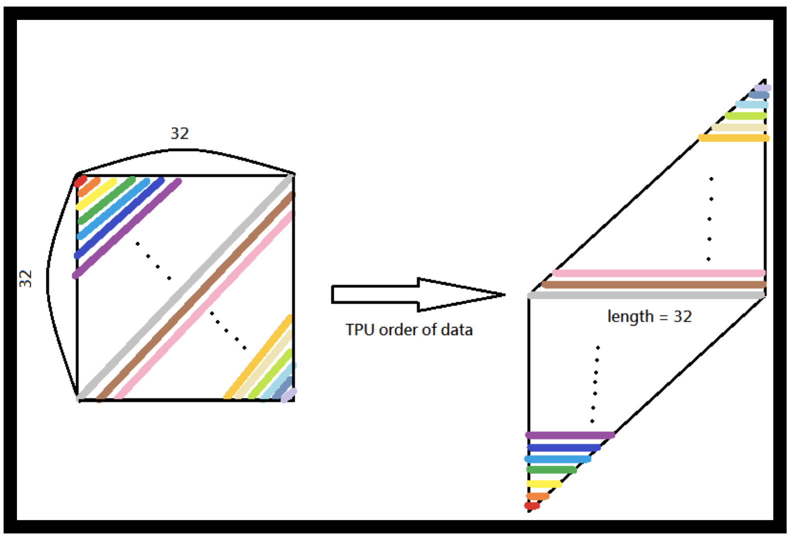
The 8×8 systolic array module instantiates an 8×8 grid of PEs with interconnections that ensure data movement in both horizontal and vertical directions. With the updated data feeding strategy, **matrix A** enters from the left, while **matrix B** is fed in from the top. This means:

- The leftmost column PEs receive the corresponding A values directly, and for every row these values shift rightward across the PEs.
- The top row PEs receive the corresponding B values directly, and for every column, these values propagate downward.

Data Feeding Strategy:
To multiply matrices A and B (such that C = A × B), the matrices must be injected into the array sequentially. In our design, matrix **A** (weights) is loaded into the array row-wise from the left, and matrix **B** (data) is loaded column-wise from the top. On the first clock cycle, the leftmost PE in each row receives the first element of matrix A, while the topmost PE in each column receives the first element of matrix B. In subsequent cycles, subsequent elements are streamed in following a **wavefront pattern**: each cycle, the next element of A's row is shifted rightward, and the next element of B's column is introduced downward. Over 8 + 8 – 1 = 15 cycles (after initial latency), the entire data set has traversed the array, and each PE has computed its portion of the output matrix.



a_{41} a_{31} a_{21} a_{11}
a_{42} a_{32} a_{22} a_{12} 0
a_{43} a_{33} a_{23} a_{13} 0 0
a_{44} a_{34} a_{24} a_{14} 0 0 0
b_{14} b_{13} b_{12} b_{11}
b_{24} b_{23} b_{22} b_{21} 0
b_{34} b_{33} b_{32} b_{31} 0 0
b_{44} b_{43} b_{42} b_{41} 0 0 0
PE



Top-Level Module (MatMul8x8_Top)

The MatMul8x8_Top module integrates the systolic array with external interfaces. It is responsible for feeding the input matrices into the array and collecting the computed results. Under the revised data feeding scheme, the top-level module presents:

- **Matrix A values:** Shifted in row-wise from the left into the array.
- **Matrix B values:** Shifted in column-wise from the top into the array.

The module includes input registers or buffers (possibly implemented with FIFO queues) that handle these directional flows. Control logic sequences these inputs so that on each clock tick, new A values (coming from the left) and new B values (descending from the top) are correctly applied. As the array operates, partial sums are accumulated by each PE. Once the computation completes, the outputs from the bottom row (or rightmost column) of PEs are collected to form the final output matrix C.

Testbench (tb_MatMul8x8)

The testbench is a non-synthesizable Verilog module used for simulation and verification. It instantiates the MatMul8x8_Top module and connects its inputs and outputs, generating a clock signal (e.g., with a 10ns period) and a reset signal to initialize the design. Under this configuration, the testbench:

- **Initializes Input Matrices:** Provides two 8×8 matrices, where matrix **A** is defined to be fed in row-wise from the left and matrix **B** is defined to be fed in column-wise from the top.
- **Stimulus Application:** After deasserting the reset, it sequences the input values into the top-level module in synchrony with the array's expected wavefront pattern.
- **Output Verification:** Once the systolic array has processed the inputs over the required cycles, the testbench captures the output matrix C, comparing each element against the expected matrix product computed via a reference model. Success and error messages (or assertions) are displayed, confirming that each computed value matches the theoretical dot-product result.

For example, if matrix A is an identity matrix and matrix B is defined with known values, the output matrix C should exactly equal matrix B (since I × B = B). Testing with varied inputs validates that the systolic array correctly computes every element of C.

Why the Systolic Array is Faster and More Efficient

1. **Dedicated Parallel MAC Units:**
   All 64 PEs work concurrently in a tightly coupled pipeline. This means that once the pipeline is filled, every clock cycle contributes directly to computation, achieving a throughput that is orders of magnitude higher for matrix multiplication tasks.

2. **Efficient Data Reuse:**
   The design leverages the fact that each element of Matrix A (fed from the left) is reused across an entire row of PEs, and each element of Matrix B (fed from the top) is reused down an entire column. This minimizes memory accesses and thereby reduces latency and energy consumption.

3. **Deterministic Latency and Low Cycle Count:**
   With a fixed pattern and no need for complex control flow, the systolic array ensures predictable performance. For example, our 8×8 design always takes roughly 22 cycles—from the first data input until the result is fully available—making it ideal for real-time and power-efficient deep learning applications.

4. **Energy Efficiency:**
   By avoiding the multiple levels of caching, dynamic scheduling, and instruction fetching inherent in CPUs and GPUs, systolic arrays require significantly less energy per operation. In deep learning, where matrix multiplications are performed repeatedly (often billions of times during training or inference), these savings translate into overall higher efficiency per watt.

Simulation Results and Discussion

After implementing the design, simulations were performed using a Verilog simulator with a clock period set (e.g., 10ns) to step through the systolic array operations. Internal signals were monitored to verify that data was flowing as expected: each PE correctly forwarded its A input rightward and its B input downward, while accumulating the respective multiply-add products.

Test Case: One representative case multiplied two 8×8 matrices with a mix of positive and negative integer entries (ensuring that signed arithmetic was exercised). For instance, if the first row of matrix A (fed from the left) and the first column of matrix B (fed from the top) contain specific test values, the resulting output element C(0,0) is computed as the dot product of that row and column. Detailed verification showed that every PE produced the correct output, with the full 8×8 result matching the expected mathematical product computed offline.

During the steady phase of operation, after the initial pipeline fill, the array produced one output per clock cycle. Compared to a sequential multiply-accumulate unit, the systolic array's parallel execution delivered the entire matrix multiplication in roughly 15 cycles (plus any additional cycles for pipeline flush), demonstrating a significant speedup. The simulation further illustrated the "wavefront" pattern of data propagation, with A values steadily moving rightward and B values descending from the top until every PE had operated concurrently.

Overall, the results confirm the design's correctness and the advantageous parallelism provided by the systolic array. The output matrix C precisely corresponds to the expected product, validating both the hardware's arithmetic operations and the revised directional data feeding strategy.

Conclusion and Future Enhancements

In this project, we successfully designed, implemented, and verified an 8×8 systolic array matrix multiplier in Verilog, drawing inspiration from the TPU architecture. With the updated data feeding approach—where matrix **A** is fed horizontally from the left and matrix **B** is fed vertically from the top—the systolic array efficiently computes the product of two 8×8 matrices using 64 parallel Processing Elements. The simulation results confirm the proper operation of the PE data propagation and MAC operations, while also highlighting the inherent speedup provided by parallel processing.

Key Findings:

- The directional feeding of matrix A from the left and matrix B from the top streamlines data entry and maximizes on-chip data reuse.
- The systolic array computes the full 8×8 matrix product in roughly 15 active cycles once the pipeline is filled, showcasing superior efficiency over sequential implementations.
- The design reinforces the concept that specialized hardware—like TPUs—can dramatically accelerate deep learning computations by harnessing parallelism and localized data movement.

Future Enhancements:

- **Support for Larger Matrices:** Scale the array (or tile multiple arrays) to handle N×N matrix multiplications, thereby processing larger networks or datasets.
- **Increased Precision or Floating Point Operations:** Upgrade from 8-bit fixed-point arithmetic to 16-bit, 32-bit, or floating-point to broaden application scenarios and improve numerical accuracy.
- **FPGA Implementation and Validation:** Synthesize the design on FPGA hardware to verify real-world performance, clock frequencies, and power consumption.
- **Enhanced Memory Interfaces:** Integrate DMA controllers or standardized bus interfaces (e.g., AXI) to enable seamless communication between the accelerator and host memory.
- **Continuous Pipelining:** Extend the design to allow back-to-back matrix multiplication tasks with minimal idle cycles, thereby emulating real-time inference scenarios in deep neural networks.

In summary, by updating the directional data feed—matrix A from the left and matrix B from the top—we have tailored the systolic array design to a commonly used convention in hardware accelerators. This adjustment, along with the robust simulation and verification, paves the way for further optimizations and real-world applications in AI hardware acceleration.

Summary of Comparison

- **CPU (Naïve vs. Optimized):**
  - **Naïve count (no caching):** Up to ~1536 memory accesses
  - **Optimized count (with caching):** ~192 global memory calls (plus many on-chip accesses at register/cached levels)
- **GPU (Naïve vs. Optimized):**
  - **Naïve (per-thread independent loads):** ~1088 global memory calls
  - **Optimized (using shared memory):** ~192 global memory calls
  - Plus, advanced coalescing and pipelining hide latency effectively.
- **TPU (Systolic Array):**
  - The systolic architecture streams in all required data once (128 calls for inputs) and writes out results (64 calls), yielding ~192 global memory calls in an ideal, fully-optimized scenario.
  - Its design emphasizes data reuse such that almost every arithmetic operation is performed on data that has already been loaded.

CPUs rely on hierarchical caches and careful software organization to minimize costly main memory accesses.

GPUs make heavy use of shared memory and coalesced accesses to load blocks of data efficiently.

TPUs are architected with systolic arrays that stream data across a grid of processing elements, ensuring that every value is reused many times on-chip before going off-chip.

Thus, if one optimizes for the hardware's strengths, an 8×8 matrix multiplication might incur roughly 192 global memory calls (for input loads and output stores) on each platform—even though the raw underlying number of arithmetic operations differs dramatically. This is one of the critical reasons why TPUs and GPUs achieve high performance for deep learning: they minimize expensive off-chip memory traffic by maximizing on-chip data reuse.