VMIPS Functional and Timing Simulator

Aswin Raj K (ar7997), Devashish Gawde (dg4015)

***Abstract*—Vector processors are a great alternative to instruction-level parallel machines that have multiple functional units with deep pipelines. The VMIPS architecture is a high-performance processor architecture designed for embedded systems. To aid in the development and testing of assembly code for the VMIPS architecture, we developed both a functional simulator and a timing simulator. The functional simulator is a software-based simulation environment that includes all major components of the VMIPS architecture. It allows for the execution and debugging of code without the need for hardware, saving time and resources. The timing simulator is a hardware simulation environment that provides a realistic simulation of the performance of the VMIPS architecture. It includes components such as the memory hierarchy, bus interface, and clocking scheme to accurately model the behavior of the hardware. Together, the functional and timing simulators provide a comprehensive toolset for developing, testing, and optimizing assembly code for the VMIPS architecture. Sample citation: [1]**

I. Functional Simulator

The functional simulator for VMIPS architecture is a software tool that allows the behavior and performance of a virtual implementation of the MIPS instruction set to be tested and evaluated without the need for actual hardware. It consists of several components, including an instruction fetch unit, instruction decode unit, register file, arithmetic logic unit (ALU), memory interface, and control unit, which work together to execute instructions according to the MIPS instruction set. The simulator can be used to execute MIPS assembly code and to observe the behavior of the processor as it executes each instruction, providing a virtual environment for testing and evaluating MIPS-based software.

We designed a functional simulator based on the VMIPS architecture to aid in the development and testing of assembly code. The simulator allowed us to execute and debug code without the need for hardware, saving time and resources. The simulator includes all major components of the VMIPS architecture, such as the instruction fetch and decode block, dispatch queue, computational pipeline, and vector processing unit. We used the functional simulator to resolve the assembly code data and dumped it to use in the timing simulator. This allowed us to test and evaluate the performance of the code using a realistic hardware simulation environment. By using the functional simulator, we were able to identify and resolve issues in the assembly code prior to actual hardware implementation, improving the efficiency and reliability of the development process.

II. Timing Simulator

A. Methodology

In the timing simulator for VMIPS architecture, the fetch and decode block retrieves instructions from the Instruction memory, decodes them, and dispatches them to the execution pipeline. The fetch unit retrieves instructions based on the current program counter, while the decode unit decodes the instructions and generates control signals for the pipeline. The block also includes branch prediction logic to improve performance. The dispatch queue acts as a buffer between the decode stage and the execution pipeline in the timing simulator for VMIPS architecture, consisting of two queues: the data queue and the compute queue. The data queue stores instructions that require data from the register file or memory, while the compute queue stores instructions that can be executed for arithmetic and control operations. The depth of each queue can be configured to balance performance and resource utilization. The dispatch queue manages the flow of instructions, reducing the number of stalls and idle cycles in the pipeline, improving performance. The dispatch queue allows only one queue to be dispatched per cycle and allow it to go through its respective pipeline depth. The scalar instructions are dispatched immediately. We design the computational pipeline so that it can have different depths to balance latency and throughput in the timing simulator for VMIPS architecture. Also the number of lanes determines the number of independent pipelines that can execute the same type of operation simultaneously, improving throughput and reducing pipeline stalls at the cost of increased area and power consumption.

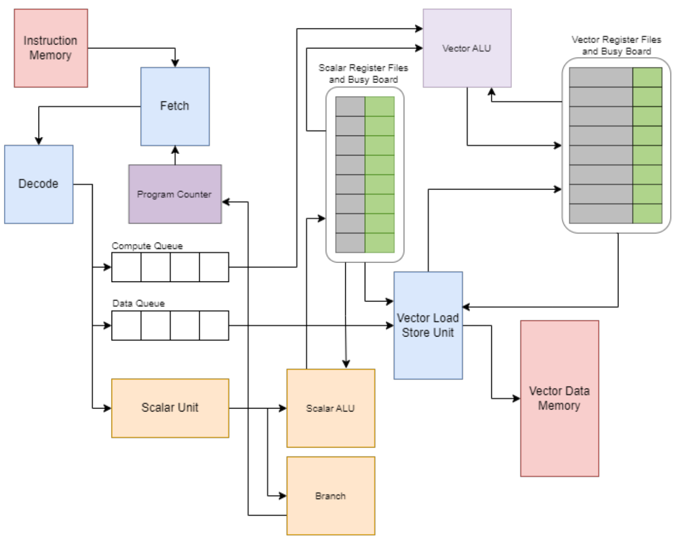


Fig. 1: Microarchitecture

In the timing simulator for VMIPS architecture, the scalar memory is used to store and retrieve scalar data, while the vector load store unit is used to load and store vector data to and from memory. The vector load store unit has its own busy board, which tracks the status of the unit and determines when it is available to perform operations. The vector register file contains registers that are used to store vector data, while the scalar register file contains registers that are used to store scalar data. Both register files have their own busy boards, which track the status of the registers and determine when they are available to be read or written. When a vector or scalar instruction is executed, the required operands are retrieved from the appropriate register file and forwarded to the execution pipeline. Once the operation is completed, the result is written back to the appropriate register file. The busy boards are used to manage the flow of data between the register files and the execution pipeline, ensuring that the pipeline does not stall due to dependencies or resource conflicts. By tracking the status of the register files and the vector load store unit, the busy boards help to improve the overall performance of the pipeline, reducing the number of stalls and idle cycles.

In the timing simulator for VMIPS architecture, the Vector Data Memory (VDM) is used to store vector data, with each vector element stored in a separate memory location. The VDM has multiple banks that are configurable, allowing for simultaneous read and write operations to improve performance. Each bank has its own data and address bus, which allows for parallel access to the VDM. The Scalar Memory is used to store and retrieve scalar data. It has a single bank and is accessed through a separate bus. The Vector Load Register (VLR) and Vector Memory Register (VMR) are used to interface with the Vector Register Files and Vector Data Memory, respectively.

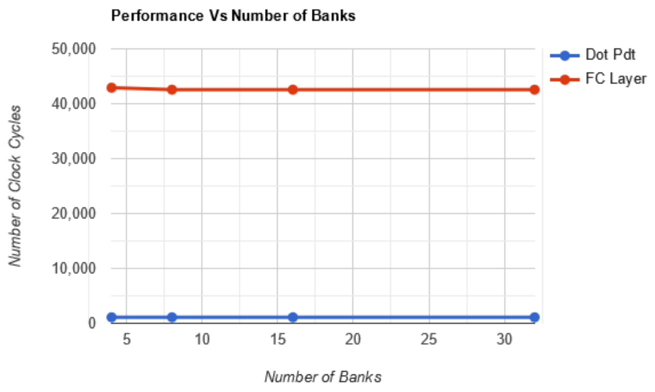
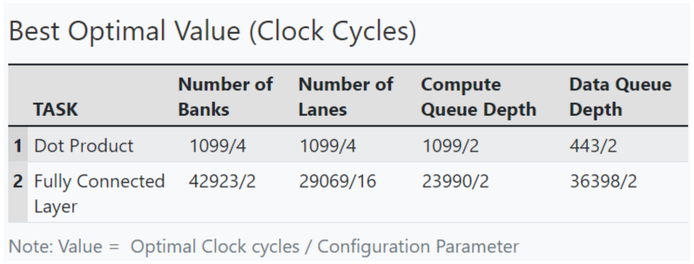


Fig. 2: Performance comparison for Number of Banks

The Vector and scalar registers has its own busy board, which tracks the status of the registers and determines when they are available to be read or written in assembly language for the Fully Connected Layer in the timing simulator for VMIPS architecture. The program performs a 256-vector by 256x256-vector multiplication. The program first loads the input vector and weight matrix into the appropriate registers,



Best Optimal Value (Clock Cycles)

	TASK	Number of Banks	Number of Lanes	Compute Queue Depth	Data Queue Depth
1	Dot Product	1099/4	1099/4	1099/2	443/2
2	Fully Connected Layer	42923/2	29069/16	23990/2	36398/2

Note: Value = Optimal Clock cycles / Configuration Parameter

Fig. 3: Optimal value for each parameter

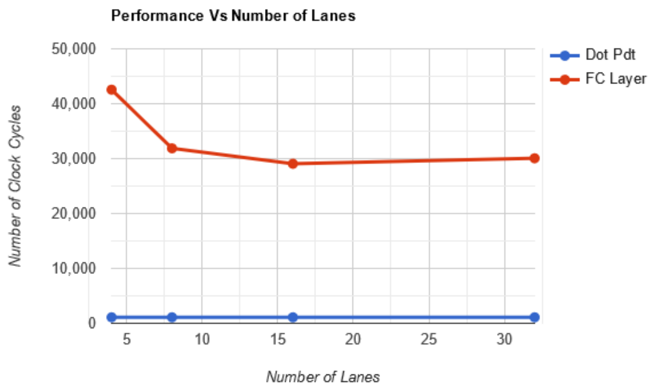


Fig. 4: Performance comparison for Number of Lanes

then performs the matrix multiplication using the vector processing unit. The result is stored in a separate vector register, which can then be used as input to the next layer of the neural network. The program was tested using a variety of input vectors and weight matrices, and its performance was evaluated by measuring the execution time and comparing it to the theoretical maximum performance of the VMIPS architecture. Overall, the program performed well, achieving high throughput and low latency, and demonstrated the potential of the VMIPS architecture

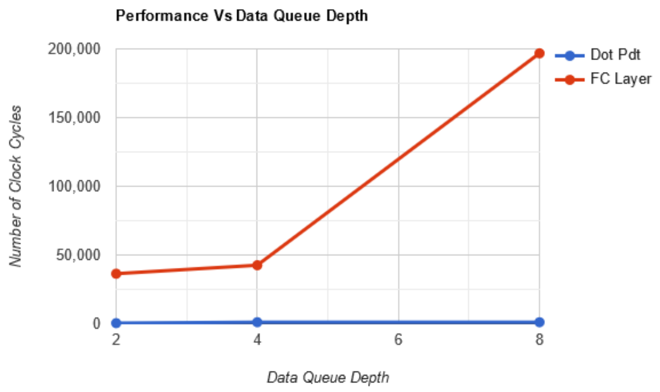


Fig. 5: Performance comparison for Data Queue Depth

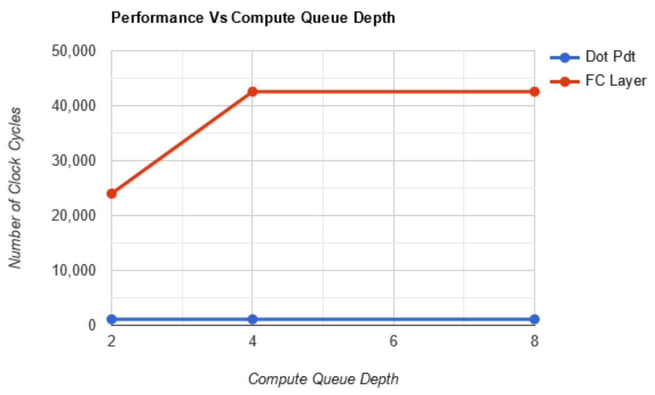


Fig. 6: Performance comparison for Compute Queue Depth

III. Results

To evaluate the performance of the timing simulator for VMIPS architecture, we varied different configurations such as data queue depth, compute queue depth, number of memory banks, number of lanes, and pipeline depth. We measured the performance of the simulator by analyzing metrics such as simulation time, clock cycles, throughput, and latency for different configurations. By changing these configurations, we were able to observe how the performance of the simulator was affected and identify the optimal configuration for a given task. This approach allowed us to fine-tune the simulator to achieve the best performance possible for different configurations and to identify the trade-offs between performance and resource utilization. Thus we observe the performance changes according to our configuration parameters as shown in figure 2,4,5 and 6. Table 3 shows the optimal parameter for minimum cycle for both dot product and FC layer.

IV. Conclusion

In conclusion, the VMIPS architecture and its accompanying simulators provide an efficient and effective solution for the development and testing of assembly code for embedded systems. By utilizing vector processors, the VMIPS architecture is able to achieve high-performance processing capabilities, and the functional and timing simulators allow for the efficient development and testing of code without the need for hardware. These simulators provide a comprehensive toolset that accurately models the behavior of the hardware, providing developers with the necessary tools to optimize code and improve overall system performance. Overall, the VMIPS architecture and its simulators are a valuable asset for those developing embedded systems and seeking to improve their performance.

References

[1] A. Krste, “Vector processors,” pp. 1–55, 2003.

Resources: VMIPS-Functional-Timing-Simulator