ECE-GY-9413 Course Project - VMIPS Timing Simulator

Anish Miryala (am12553) , Amuktha Kotamarthy (ak9327)

***Abstract*—In this project, we implemented a Timing Simulator for a VMIPS Processor with given specifications. The timing simulator is based on the functional simulator which is the previous part of the project. Overall, it can be seen from the results that both the simulators work according to the given specifications.**

I. Introduction

This report presents the implementation of a functional and timing simulator for a given Instruction Set Architecture (ISA) specification of a VMIPS Processor using Python. The report covers the implementation of the functional simulator, the testing of the simulator with a dot product, and the implementation of a timing simulator using the functional simulator. Additionally, the report includes a description of the microarchitecture followed, including its major stages, the busy board, and memory units. Finally, the report presents the execution of a Fully Connected Layer program to showcase the performance of the VMIPS architecture.

A. Functional Simulator

The first part of the project involved implementing a functional simulator for the given ISA specification of the VMIPS Processor. The simulator was written in Python and followed the ISA requirements to adjust the architectural state and execute instructions. The simulator was capable of simulating the execution of assembly-formatted instructions on Scalar and Vector Data memory. After the instructions were executed, the simulator output the final states of the data memories, scalar register file, and vector register file. The dot product was used to test the performance of the functional simulator.

B. Timing Simulator

A timing simulator is developed to assess the performance of the microarchitecture of the processor. The simulator accounts for the pipeline stages, functional units, and other hardware elements that affect how quickly an instruction is executed. Based on the model and inputs given, the simulator estimates how long each instruction will take to execute. Timing simulators are helpful in the research and development of computer architecture, as they allow designers to assess the performance of various microarchitectures or instruction sets without having to create actual hardware. Timing simulators can also be used for software testing and development by giving programmers information about the performance characteristics of their code on various microarchitectures.

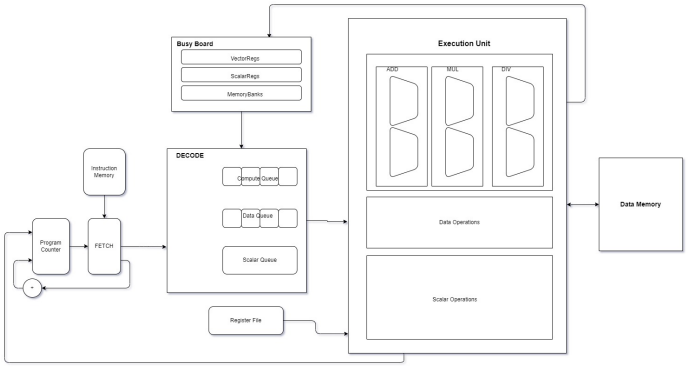

Fig. 1. Micro architecture

II. Micro architecture

In order to balance throughput and latency in the timing simulator for VMIPS architecture, we construct the computational pipeline to have flexible depths and multiple lanes. The number of lanes also defines the number of distinct pipelines that may perform the same sort of function simultaneously, increasing throughput and eliminating pipeline stalls at the expense of requiring more area and power.

The microarchitecture used in this project, shown in Figure 1, consists of three major stages: Fetch, Decode, and Execution unit. The Fetch is the first operation of the microarchitecture, where a single instruction is held at this stage from the Instruction Memory (IMEM). From here, we go to further instructions with the help of the program counter (PC). In this stage, only branch instructions are resolved.

From Fetch, we move to the Decode stage, which has three queues: Compute Queue, Data Queue, and Scalar Queue. The Compute Queue handles all vector operations other than the load-store operations, the Data Queue handles all load-store operations, and the Scalar Queue handles all scalar operations. In this stage, the instructions get decoded and get added to their respective queues ready for dispatch in the next clock cycle. Depending on the register value from the busy board, the instructions get stalled or, get added to the queues.

The third stage is Execution, which is the most critical part of the microarchitecture, where all vector and scalar operations like arithmetic and load-store operations are handled. It consists of three sub-components; Compute, Data, and Scalar execution units respectively. The parameters of the execution units can be configured using the 'config' file in the project. For example, the Compute unit can be configured for different

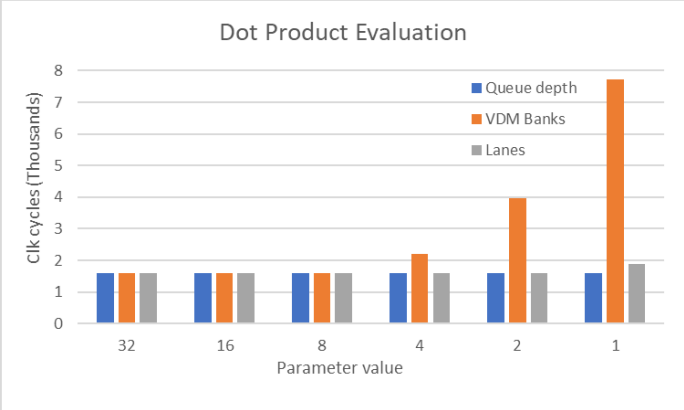

Fig. 2. Performance Evaluation of Dot Product

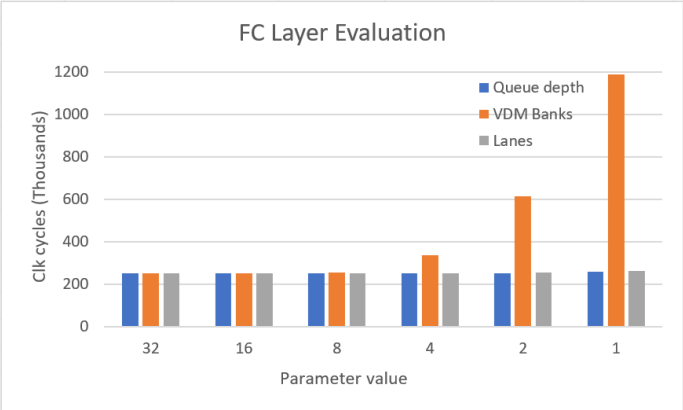

Fig. 3. Performance Evaluation of Fully Connected Layer

pipeline depths and lanes for each of add, multiply, and division operations separately.

Busy Boards and Memory Units: Two busy boards are used in the microarchitecture, one for vector registers and one for scalar registers. The purpose of the busy board is to determine which vector or scalar register is occupied and which of them are free for instruction to get dispatched. If the registers are occupied, the busy board is set to high (1), and if they are free, the busy board is set to low (0).

The Memory unit consists of two parts; Scalar Memory and Vector Memory. The vector memory is divided into multiple banks (number of banks can be configured using the config file) and each bank has a bank busy time for which, no operations can be performed on that bank. This is also implemented using a separate busy board for memory which indicates whether a bank is free for data read or write. Whenever a data operation is performed, the value in the busy board is set high till the bank busy time.

III. Experimentation Methodology

The Functional simulator, designed previously, has been used to generate a resolved code that can be fed into the timing simulator. The timing simulator estimates the time taken for all the instructions to get executed based on the model and inputs given. The timing simulator was used to assess the performance without the need for actual hardware. The program's parameters were varied, including the number of lanes and pipeline depths, to compare the number of clock cycles for different combinations.

IV. Performance Evaluation

In this section, we will evaluate the results of our dot product and also a fully connected layer.

A. DotProduct

The dot product was implemented successfully using the functional simulator between two vectors of size 256. The resolved code suitable for the timing simulator has been generated and executed successfully. Figure 2 shows the plot of timing data with different values of parameters as specified.

B. Fully Connected Layer

The timing simulator was used to execute a Fully Connected Layer program that involves the multiplication of a vector (size 256) by a Matrix (size 256x256). The input vector and weight matrix were loaded into the memory before using the functional simulator. The program was tested using various input vectors and weight matrices, and the execution time was compared with various parameter values. The program delivered high throughput and low latency overall, showcasing the promise of the VMIPS architecture.

V. Conclusion

In conclusion, the project involved implementing a functional simulator for a given Instruction Set Architecture (ISA) specification of a VMIPS Processor using Python. We were able to successfully implement the simulator to adjust the architectural state and execute instructions in accordance with the ISA requirements. We tested the simulator by performing a dot product and executing a fully connected layer to evaluate its performance.

For the Dot Product, the performance improves by 4.8x when the banks are changed from 1 to 8 and then remain the same thereafter. Also, the cycles increase by 17% when the number of lanes are dropped from 8 to 1, which is not that significant.

For the Fully Connected Layer, the performance improves rapidly at the beginning when the banks are doubled starting from 1 bank to 4 banks (almost 1.9x), and then does not change much after 8 banks. Doubling the lanes accounts for only a 2% drop in cycles at every stage, which is not a healthy return.

After analyzing the results, it can be concluded that the Number of Lanes and the value of Compute and Data Queue does not affect the performance significantly. In contrast, the effect of the Number of Memory Banks can be seen clearly.

References

Project link: VMIPS-Func-Timing-Simulator