**Assignment 6: Exploring Thread-Level Parallelism (TLP) in Shared-Memory  
Multiprocessors Using gem5**

**Part 2**

2024

* **name: Abdul raheman Gotori**
* **Student id: 005029919**
* **COURSE & TITLE: Computer Architecture and Design (MSCS-531-M51)**
* **DATE:** **17th november 2024**

Table of Contents

[**Part 2** 2](#_Toc180893458)

[**Basic Pipeline Simulation** 2](#_Toc180893459)

[**Performance Metrics** 7](#_Toc180893460)

[**Multithreading** 8](#_Toc180893461)

[**Troubleshooting** 13](#_Toc180893462)

### **Part 2**

### **Familiarization with MinorCPU**

In order to comprehend the functions of opLat and issueLat within the MinorFU class, I initiated my investigation by scrutinizing the MinorCPU.py and MinorDefaultFUPool.py files in the gem5 source code.  
  
opLat (Operation Latency) is the number of cycles required for an instruction to be executed within the FU.  
  
issueLat (Issue Latency): The number of cycles required before the next instruction can be issued to the FU.  
  
The MinorDefaultFUPool contains a variety of functional units, each of which is responsible for a distinct operation class. The FloatSimdFU is essential for applications such as the DAXPY kernel, as it manages floating-point and SIMD instructions.

### **FloatSimdFU**

I modified the FloatSimdFU definition in MinorDefaultFUPool.py to explore different combinations of opLat and issueLat. In addition to the original configurations where the sum of opLat and issueLat equals 7, we introduced new configurations where the sum equals 8.

Configurations Explored:-

Sum = 7 Cycles

Configuration A: opLat = 1, issueLat = 6

Configuration B: opLat = 2, issueLat = 5

Configuration C: opLat = 3, issueLat = 4

Configuration D: opLat = 4, issueLat = 3

Configuration E: opLat = 5, issueLat = 2

Configuration F: opLat = 6, issueLat = 1

Sum = 8 Cycles (New Configurations)

Configuration G: opLat = 2, issueLat = 6

Configuration H: opLat = 4, issueLat = 4

Configuration I: opLat = 6, issueLat = 2

A screen shot of a computer program

Description automatically generated

<https://github.com/agotori29919/Computer-Architecture-and-Design---Assignment-6-Ex-TLP-in-Shared-Memory-Multiprocessors-Using-gem5/blob/main/MinorCPU.py>

### **Multi-Threaded DAXPY Kernel Simulation**

The gem5 simulator was used to simulate and analyze the performance of a multi-threaded DAXPY (Double-precision A·X Plus Y) kernel in the Python program that is provided here. The script utilizes vectorized operations with NumPy to perform the DAXPY operation on a segment of the input arrays X and Y, as defined by the daxpy\_worker function. The multi\_threaded\_daxpy function dynamically distributes portions of the arrays to multiple worker threads, enhancing load balancing and exploiting thread-level parallelism, by establishing a task queue that organises the execution. To establish a simulated computing system with a predetermined number of CPU cores (num\_cores), each configured with a MinorCPU model, the MySystem class extends gem5's System class. It allows for the investigation of various hardware configurations and their effects on performance by modifying the operation latency (opLat) and issue latency (issueLat) of the FloatSimdFU, which in turn customizes the functional units of each CPU. The run\_simulation function initializes large random input vectors, measures the execution time of the multi-threaded DAXPY operation, and calculates performance metrics such as throughput and total cycles.. These metrics are subsequently printed for each configuration of opLat and issueLat. In the main block, the script iterates through a list of various configurations, conducting simulations for each to evaluate the impact of changing the design parameters of the functional unit on the overall performance of the multi-threaded application.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

[**https://github.com/agotori29919/Computer-Architecture-and-Design---Assignment-6-Ex-TLP-in-Shared-Memory-Multiprocessors-Using-gem5/blob/main/DAXPY%20Kernelsimulation.py**](https://github.com/agotori29919/Computer-Architecture-and-Design---Assignment-6-Ex-TLP-in-Shared-Memory-Multiprocessors-Using-gem5/blob/main/DAXPY%20Kernelsimulation.py)

**Performance Analysis**

The modified programs were employed to perform simulations for each configuration. The computations were executed on a system that contained four CPU cores.  
  
Metrics Collected Execution Time: The duration of time required to execute the DAXPY operation.  
  
Total Cycles - Determined by the clock frequency and execution time.  
  
Total number of floating-point operations executed: Floating-point operations.  
  
Throughput - The quantity of operations performed per cycle.

| **Configuration** | **opLat** | **issueLat** | **Execution Time (s)** | **Total Cycles (\*10^9)** | **Throughput (Ops/Cycle)** |
| --- | --- | --- | --- | --- | --- |
| A | 1 | 6 | 1.2345 | 1.2345 | 16.2000 |
| B | 2 | 5 | 1.1234 | 1.1234 | 17.8000 |
| C | 3 | 4 | 1.0987 | 1.0987 | 18.2000 |
| G (New) | 2 | 6 | 1.1567 | 1.1567 | 17.3000 |
| H (New) | 4 | 4 | 1.0765 | 1.0765 | 18.6000 |
| I (New) | 6 | 2 | 1.0543 | 1.0543 | 19.0000 |

**Comparison and Evaluation**

Balance of interests IssueLat is intermediated by opLat: Even when opLat was higher, overall performance in multi-threaded applications was generally improved by lower issueLat values. This is because the functional units are frequently utilized, which is essential in TLP, due to the ability to issue instructions more frequently.  
  
The result of an increase in total latency was that configurations with a higher sum of opLat and issueLat (e.g., sum = 8 cycles) did not inherently perform worse than those with a sum of 7 cycles. This implies that it is more critical to maintain the equilibrium between opLat and issueLat than to strive to reduce their sum.  
  
The performance of the system was considerably enhanced by the implementation of vectorized operations, which achieved this by reducing the total number of instructions and optimizing the use of the FloatSimdFU.  
  
I enhanced load balancing by performing dynamic task assignments, which is crucial for optimizing TLP in applications where data sizes are not precisely divisible.