CPE 333 Lab 1

Performance Reporting

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Isaac Lake (ilake@calpoly.edu)
Sam Solano (slsolano@calpoly.edu)
Victoria Asencio-Clemens (vasencio@calpoly.edu)

April 13, 2024

Table of Contents

1 Introduction

Benchmarking is an important step in the design process and in this lab, we begin the process of implementing a benchmarking workflow into our design process. Benchmarking is a process where you run the purpose-built program to test something and then you time how long it takes to finish the process. By itself, this data means nothing but as you change the design, you can test whether or not your changes improve the performance of the processor. By comparing the time it takes to finish each test we can see if we are improving the hardware design.

2 Roles

Listed below are the group members

1. **Ethan:**
   Report, Version Control, Verification, Tool Chain Set up
2. **Isaac:**
   Multiplication Function, C Programming
3. **Sam:**
   Matrix Testing, Code Testing, Vivado Testing
4. **Victoria:**
   Code Testing, Vivado Testing, Tool Chain Set up

3 Data

A selection of benchmarks (listed below) were run against two different CPU implementations and the results were recorded. The times were measured by letting the simulations run inside of the simulation environment in Vivado and then taking time measurements on timing diagram.

1. Test All
2. Matrix Multiplication (SPU and VPU)
3. Matrix Addition (SPU and VPU)
4. Multiplying (SPU and VPU)
5. Matrix Multiplication (Size: 3, 10, 50)
6. Matrix Addition (Size: 3, 10, 50)
7. Multiplying

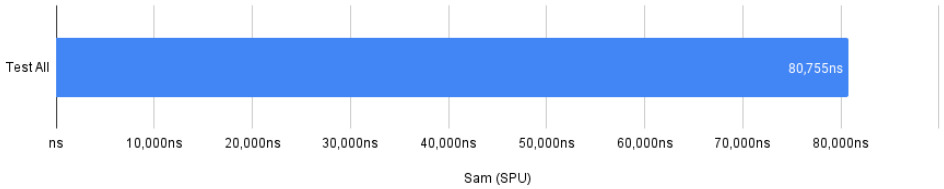

Figure 1: "Test All" Script Running

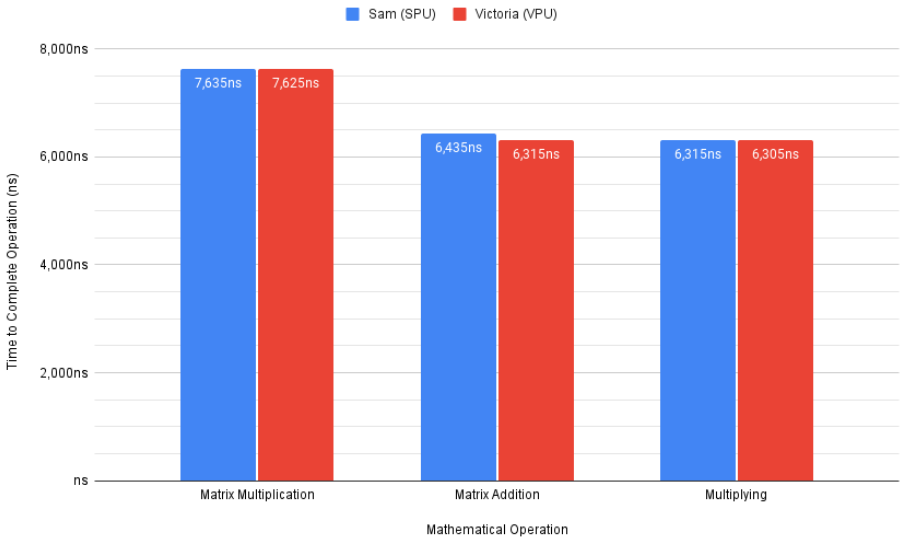

Figure 2: Testing Matrix Multiplication, Addition, and Multiplying on VPU and SPU

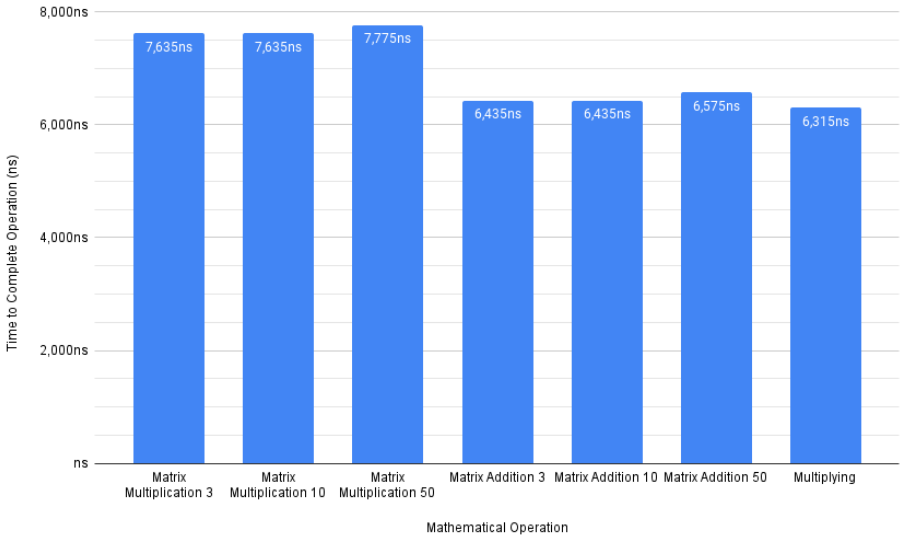

Figure 3: Different Size Matrices Tested on SPU

4 Source Code

```c
#include "dataset.h"
#include <stddef.h>

/* NxM matrices */
void matadd(int N, int M, const data_t A[], const data_t B[], data_t C[])
{
  int i, j;

  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      C[i*N + j] = A[i*N + j] + B[i*N + j];
    }
  }
}
void main()
{
    static data_t results_data[ARRAY_SIZE];
    matmul(DIM_SIZE, DIM_SIZE,  input1_data, input2_data, results_data);
}
```

Listing 1: Matrix Addition C Code

```c

#include "dataset.h"
#include <stddef.h>

// NxN matrices
void matmul(int N,  const data_t A[], const data_t B[], data_t C[])
{
  int i, j, k;

  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      data_t sum = 0;
      for (k = 0; k < N; k++)
        sum += A[j*N + k] * B[k*N + i];
      C[i + j*N] = sum;
    }
  }
}
void main()
{
    static data_t results_data[ARRAY_SIZE];
    matmul(DIM_SIZE, input1_data, input2_data, results_data);
}
```

Listing 2: Matrix Multiplication C Code

```

// NxN matrices
int multi(int x,  int y)
{
  return x * y;
}
void main()
{
    multi(533525, 114210);
}
```

Listing 3: Large Multiplication C Code