Riley Taylor

Dr. Jamieson

4/30/2022

Advanced Assembly Badge

**Part 1: Optimization Comparison MIPS**

Each comparison between MIPS and ARM simulation uses the same input data in static memory or appropriate registers.

1. Unoptimized Matrix Multiplication vs. Optimized Matrix Multiplication

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

1. Unoptimized Matrix Scalar Multiplication vs. Optimized Matrix Scalar Multiplication

Graphical user interface

Description automatically generated Graphical user interface, application

Description automatically generated

1. Unoptimized GCD vs. Optimized GCD

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

1. Unoptimized Pop Count vs. Optimized Pop Count

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

1. Unoptimized Int to String vs. Optimized Int to String

Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generated

1. Unoptimized Bubble Sort vs Optimized Bubble Sort

Graphical user interface

Description automatically generatedGraphical user interface, application

Description automatically generated

**Part 2: Optimization Comparison ARM**

1. Unoptimized Matrix Multiplication vs. Optimized Matrix Multiplication

A picture containing graphical user interface

Description automatically generatedA picture containing text

Description automatically generated

1. Unoptimized Matrix Scalar Multiplication vs. Optimized Matrix Scalar Multiplication

A picture containing text

Description automatically generatedA picture containing text

Description automatically generated

1. Unoptimized GCD vs. Optimized GCD

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. Unoptimized Pop Count vs. Optimized Pop Count

Graphical user interface, text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. Unoptimized Int to String vs. Optimized Int to String

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

1. Unoptimized Bubble Sort vs Optimized Bubble Sort

Text

Description automatically generated with medium confidence Graphical user interface, text, application

Description automatically generated

The following is a table that compares the instruction counts of the optimized programs in both MIPS and ARM. Both programs allowed to see the exact number of instructions executed using tools provided by the software.



**Further Analysis of Individual Programs, MIPS, and ARM**

The ARM simulation I used is named ARM generic, this means that the simulator does not require hardware to test ARM instructions and outputs. The simulator is not as convenient as MIPS because ARM only allows 12 registers compared to MIPS’ 22 registers (including the $a registers). ARM also has a reduced instruction set. For example, the ARM simulator does not have a way to properly exit the program. If I were using ARM for the Raspberry Pi, then there is a system interrupt that allows the program to terminate, but in the simulator, to maintain register data, it is required to pop the program counter. Overall, MIPS was easier to work with than ARM when working with these benchmarks.

Matrix multiplication was similar in instruction size. The MIPS architecture shows to be an improvement in execution time over the ARM architecture. This can be explained by ARM not having sufficient registers to efficiently carry out matrix multiplication. ARM requires, for my program, three additional registers to be freed to build the matrices, so this adds additional instructions in the matrix builder loop. MIPS has a more registers than needed to execute the program. I did not need to reassign any registers, so the matrix builder required less instructions per loop, resulting in MIPS being more efficient at matrix multiplication.

Matrix scalar multiplication was also close, like matrix multiplication. ARM has the better instruction count. Unlike matrix multiplication, I did not need to reuse registers in the ARM simulator. Most of the extra instructions in the MIPS matrix multiplication program come from the malloc command, as the ARM simulator I used does not require or have instructions to allocate the memory in the heap. ARM allows the heap to be populated outside of the static memory by using a global pointer and storing information to it.

GCD had relatively low instruction counts in both MIPS and ARM. MIPS is barely more efficient than ARM, I would even argue that the difference was negligible. MIPS pulled ahead slightly because I added a NOP to the subtraction label that allowed the program to be pipelined.

Pop count has the worst disparity in instruction count between the two architectures. The ARM simulator I used did not have a divides instruction, so I needed to create my own division label using add and subtraction instructions. I could have inserted the label inside the loop itself, but after trying that it did not improve my program’s instruction count. Since the benchmark had the exit condition after 32 loops, MIPS saved four additional instructions per loop compared to ARM. Any program or benchmark that would require division, MIPS will always perform better over the simulated ARM.

Integer to string, like the matrix scalar multiplication program, had less instructions than MIPS for the exact same reason. Most of the extra instructions in the MIPS matrix multiplication program come from the malloc command, as the ARM simulator I used does not require or have instructions to allocate the memory in the heap. ARM allows the heap to be populated outside of the static memory by using a global pointer and storing information to it.

Bubble sort for both of my optimized programs did not use the heap to store the memory because the benchmark required that the static memory is manipulated to sort the integers. By the instruction counts, ARM was more efficient than MIPS. The structure of the code is practically identical when comparing the two.

MIPS and ARM both have benefits and downfalls. MIPS shines when a lot of registers are required to store data, as MIPS provides more registers to use than ARM. Also, MIPS has convenient instructions compared to ARM, including the division function and pointers. ARM on the other hand handles the malloc operation easily, it allows the user to use a global pointer to point to the place in memory the user wants the heap to be allocated. MIPS on the other hand requires allocation of memory to use and store data on the heap. Overall, I think MIPS is more practical and easier to understand than ARM, but ARM allows for large amounts of data storage that can be useful if the program requires it.

**Works Cited**

ARM simulator: <https://cpulator.01xz.net/?sys=arm>