ECSE 324 - Lab 4 Report

Alexander Liu

April 12th, 2024

Part 3: Performance Analysis A

Code Overview

My subroutine *GoL_draw_grid_ASM* leverages the *VGA_draw_line_ASM* subroutine (which leverages *VGA_draw_point_ASM*) to draw the game grid. *VGA_draw_line_ASM* is strictly configured to draw horizontal and vertical lines only. The colour for the lines is passed to *GoL_draw_grid_ASM* in A1. I start off by instantiating my coordinates (x1, y1) and (x2, y2) which will be subsequently passed to the line-drawing subroutine. Then, I have 2 loops in the body of my grid-drawing subroutine with the first loop to cover all vertical lines and the second loop to cover all horizontal lines. There is some setup required to transition from one loop to another which occurs upon exiting the first loop.

Performance Analysis

Before making any analysis on performance, it is important to establish definitions. In this report, a data memory access will refer to an explicit load or store instruction which excludes fetching instructions as memory accesses. We define data memory access this way because we are more concerned with the different subroutines' runtimes. There are 3 subroutines that we must analyze to gauge performance: *GoL_draw_grid_ASM*, *VGA_draw_line_ASM* which is called inside the body of the encapsulating subroutine *GoL_draw_grid_ASM*, and *VGA_draw_point_ASM* which is called within *VGA_draw_line_ASM*. The easiest way to account for the number of data memory accesses performed by all 3 subroutines to draw the grid is through breakpoints. The methodology employed here is to insert a breakpoint at the line where I call the subroutine *GoL_draw_grid_ASM* and another breakpoint immediately after on the next line. The images below showcase where the breakpoints were inserted explicitly as well as provides the relevant data before and after the execution of *GoL_draw_grid_ASM*.

Figure 1. Breakpoints inserted in the program's disassembly

Counter	Value
Exec. instructions	1692923
Data loads	230607
Data stores	307404
Simulator run time (ms)	495
Simulator core run time (ms)	466
Simulated MIPS	3.625

Counter	Value
Exec. instructions	1850445
Data loads	252102
Data stores	336019
Simulator run time (ms)	549
Simulator core run time (ms)	519
Simulated MIPS	3.559

Figure 2. Counter values before and after calling *GoL_draw_grid_ASM* (respectively)

From figures 2 and 3, we can perform a subtraction between the number of data loads and sum the difference with the difference between the number of data stores. Doing this operation, we get [252102 - 230607] + [336019 - 307404] = 50110 data memory accesses in total. Moreover, the number of instructions executed by calling *GoL_draw_grid_ASM* is 157522 and the runtime of the subroutine is 549 – 495 = 54 ms. This yields a data memory access to instruction executed ratio of 0.3181, meaning 31.81% of all instructions executed are data memory accesses. Overall, we can confidently say the most time is spent inside *VGA_draw_point_ASM* since we need to call it to write to every pixel in every line in the grid. We have 15 vertical lines and 11 horizontal lines. Each vertical line has 240 pixels, and each horizontal line has 320 pixels. This means that we are calling *VGA_draw_point_ASM* for a total of 7120 times while only calling *VGA_draw_line_ASM* 26 times and *GoL_draw_grid_ASM* once. Hence, the most total time spent would be within the *VGA_draw_point_ASM* subroutine. If the time required to perform a data memory access were to increase, then we can expect the runtime of the subroutine to suffer consequently. More precisely, since 31.81% of instructions are data memory accesses, we can expect a somewhat substantial increase in runtime.