EE4930 Advanced Embedded Systems

Section 011, Winter 2019/20

Draven Schilling

1-4-20

Lab 3 – Compiler Optimization

# Objectives

The objective for this lab was to experiment with the compiler optimization settings and see the results of both weighting execution speed more heavily, and code size more heavily. By writing “bad” code that has much room for optimization, how well can the compiler generate a binary that is codemore efficient.

# Description

In this lab I created some convoluted code that is intended to be inefficient and have room for optimizations. I created nested loops with unnecessary variables, if/else statements that could be simplified, multiple minimal method calls, and used unnecessarily large variable types. Furthermore, I Initialized one GPIO port as a GPIO out and toggled the output (1/2 period) every loop cycle such that I could later use an oscilloscope to measure the execution time of the main loop. I then ran the code with various optimization settings to see the changes in both code size and speed (length of main loop) and recorded the results in the provided excel spreadsheet.

# Conclusions

Everything in this lab was relatively straightforward. It was interesting to change my viewpoint and intentionally concentrate on writing inefficient code, but overall, with the suggestions provided in the lab handout, it was not too difficult. Recording the data was easy once you knew where to look it was rinse and repeat for each optimization setting. I found some of the results to be interesting; specifically, there was once optimization setting where both speed and size increased meaning the results were worse than no optimization at all, I’m not entirely sure why but it’s an interesting observation. Aside from that, it seems like the best overall results came from the max level of optimization which is somewhat expected.

# Attachments

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EE4930 |  |  |  |  |  |  |  |  |  |  |  |
| Lab 3 Data |  |  |  |  |  |  |  |  |  |  |  |
| Name: Draven Schilling | |  |  |  |  |  |  |  |  |  |  |
| Date: 1/4/20 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Optimization setting | size/speed setting | Module code size | Percent change | Total code size | Percent change | Module ro data size | Total ro data size | Module rw data size | Total rw data size | run time | Percent change |
| OFF | 0 | 1270 |  | 2350 |  | 228 | 261 | 8 | 3112 | 0.467 |  |
| 0 | 0 | 942 | -25.8% | 2022 | -14.0% | 228 | 261 | 8 | 3122 | 0.304 | -34.9% |
| 2 | 0 | 770 | -39.4% | 1850 | -21.3% | 228 | 261 | 8 | 3112 | 0.27 | -42.2% |
| 4 | 0 | 318 | -75.0% | 1394 | -40.7% | 228 | 255 | 0 | 3104 | 0.221 | -52.7% |
| 0 | 2 | 1026 | -19.2% | 2106 | -10.4% | 228 | 261 | 8 | 3122 | 0.293 | -37.3% |
| 2 | 2 | 834 | -34.3% | 1914 | -18.6% | 228 | 261 | 8 | 3112 | 0.267 | -42.8% |
| 4 | 2 | 342 | -73.1% | 1486 | -36.8% | 228 | 255 | 0 | 3104 | 0.209 | -55.2% |
| 0 | 3 | 1386 | 9.1% | 2490 | 6.0% | 228 | 261 | 8 | 3112 | 0.653 | 39.8% |
| 2 | 3 | 1062 | -16.4% | 2142 | -8.9% | 228 | 261 | 8 | 3112 | 0.265 | -43.3% |
| 4 | 3 | 348 | -72.6% | 1492 | -36.5% | 228 | 255 | 0 | 3104 | 0.108 | -76.9% |
| 0 | 5 | 1512 | 19.1% | 2616 | 11.3% | 228 | 261 | 8 | 3112 | 0.656 | 40.5% |
| 2 | 5 | 1180 | -7.1% | 2260 | -3.8% | 228 | 261 | 8 | 3112 | 0.267 | -42.8% |
| 4 | 5 | 348 | -72.6% | 1472 | -37.4% | 228 | 255 | 0 | 3104 | 0.1 | -78.6% |

**Source Code:**

\*see Comments for explanation on why each block could be optimized

// main.c

// Runs on MSP432

// Lab 3 - Compiler Optimization

// Draven Schilling

// 12/17/19

// Runs some VERY inefficient code to time the duration of the main loop.

// toggles a GPIO output every main loop cycle

// The goal is to see the effects of compiler optimization

// which will be shown as a shorter period on the GPIO out signal.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** "msp.h"

**void** **init\_gpio**();

**void** **nestedFunct**(**int** times);

**void** **innerNestedFunct**(**int** times);

**void** **innermostNestedFunct**();

**int** **dummyFunct**(**int** length);

**void** **main**(**void**)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

init\_gpio();

**while**(1){

P5->OUT = P5->OUT^1; // toggle output (1/2 period will be program duration)

nestedFunct(10);

}

}

// Setup P5.0 as GPIO output.

**void** **init\_gpio**()

{

// P5.0

P5->SEL0 &= ~0b1; // use GPIO function

P5->SEL1 &= ~0b1;

P5->DIR |= 0b1; // make output

P5->OUT &= ~0b1; // setup output low to start

**return**;

}

/\* create deep nested short functions

\*

\* the compiler should be able to optimize how these short functions are called

\* by expanding them.

\*/

**void** **nestedFunct**(**int** times){

**int** i = 0;

**for**(i = 0; i < times; i++)

innerNestedFunct(times);

}

**void** **innerNestedFunct**(**int** times){

**int** i = 0;

**for**(i = 0; i < times; i++)

innermostNestedFunct(times);

}

**void** **innermostNestedFunct**(**int** times){

**int** i = 0;

**for**(i = 0; i < times; i++){

**int** dummy = dummyFunct(times);

}

}

// Highly inefficient calculation method

**int** **dummyFunct**(**int** length){

**int** i = 0;

/\*

\* - The compiler should be able to simplify the number of steps in the calculation.

\* - Also I used many multiples of 2 such that bit shifting would be more optimal than multiplication

\* - I used longs even though the range of data is much smaller. to see if it may optimize the allocation type

\* - I created a lot of dummy variables which should be optimized out.

\* - I also went through many nested if statements which should be optimized to a single if.

\*/

**long**\* dummy\_ptr = **malloc**(**sizeof**(**long**)\*length); // using longs unnecessarily

**for**(i = 0; i < length; i++){ // ++i is generally more efficient

dummy\_ptr[i] = (((i + 4)\*96/40)%10)+(i>0?dummy\_ptr[i-1]:64); // inefficient arithmetic statement

**int** j = 4, k = 9, l = 0, m = 7, n = 3, o = 1, p = 5; // repeatedly create unused variables

// Some compounded if statements (will always go through all given 5 < length < 11

**if**(length > 2)

**if**(length > 5)

**if**(length < 12)

**if**(length < 11)

dummy\_ptr[i] = dummy\_ptr[i] % 13;

}

**int** ret = (length = 0 ? 0 : (**int**)dummy\_ptr[length-1]);

**free**(dummy\_ptr);

**return** ret;

}