ECE 561 – Project 2 Optimization for

Fast Code

# Introduction

Embedded systems need not only be correct in their work, but also be as fast as possible while maintaining this correctness. Most of the times this speed comes at the cost of precision. The goal of this project is to try and optimize the given code to get the best possible speed, while maintaining a tradeoff between the speed, precision and accuracy of the calculations. There are many possible optimizations that can be done. They are listed in the order they were done. Also the time taken for each of the iteration is given after each optimization.

The profiling code used taps the code every 1ms to check where the PC is and populates a data-structure called Regions. By analyzing this data-structure, we can see where the code is spending most of its time and work on optimizing the corresponding area to reduce the time taken.

The operation that has to be optimized is reading the accelerometer to detect the orientation of the board, calculating the roll and pitch, using these to compensate the magnetometer readings, and calculate the heading of the board, to finally get the complete orientation of the Freedom KL25Z board in space.

# Optimizations and Results

## Initial Code

The initial code had a total execution time of 10s for 10000 iterations of the operation that is to be optimized. The statistics for this are as follows.

Total Time Taken for 10000 iterations: 10.33s

Time Taken for each iteration: 1033µs

Top 5 functions with the most number of profile ticks taken:

|  |  |
| --- | --- |
| dmul | 2751 |
| i2c\_repeated\_read | 1960 |
| i2c\_read\_setup | 1120 |
| double\_epilogue | 1104 |
| ddiv | 766 |

## Optimization 1: Using the --fpmode=fast option

This optimization forces the compiler to use an aggressive way of floating point calculations. This does reduce the accuracy of the operation to a small amount, but the increase in speed is very significant. This tradeoff is a simple one and we can see the difference in the time taken for the same calculations in the following statistics.

Total Time Taken for 10000 iterations: 6.84s

Time Taken for each iteration: 684µs

Top 5 functions with the most number of profile ticks taken:

|  |  |
| --- | --- |
| dmul | 1695 |
| i2c\_repeated\_read | 1595 |
| i2c\_read\_setup | 991 |
| \_\_ARM\_scalbn | 464 |
| fdiv | 281 |

## Optimization 2: Using Optimization Level O3

This step asks the compiler to use the highest level of optimization available. This uses lots of optimizations that can reduce the time taken, such as calculating dependencies between instructions and reordering them to reduce CPU stall cycles, etc.

Total Time Taken for 10000 iterations: 6.815s

Time Taken for each iteration: 681.5µs

Top 5 functions with the most number of profile ticks taken:

|  |  |
| --- | --- |
| dmul | 1607 |
| i2c\_repeated\_read | 1547 |
| i2c\_read\_setup | 1047 |
| \_double\_epilogue | 464 |
| \_dsqrt | 367 |

## Optimization 3: Using pre-calculated values for the constant 180/π

The compiler generally detects constant values within statements and optimizes the code to use this predefined constant values. But sometimes this optimization might not happen and this causes the constant value to be calculated every time the statement is executed. In our case, the conversion of angles from radians require it to be multiplied by the constant 180/π. By hand-calculating this value and using as a pre-processor define directive, we can make sure that we have only one multiplication done within the statement instead of a multiplication and one division.

Total Time Taken for 10000 iterations: 6.784s

Time Taken for each iteration: 678.4µs

Top 5 functions with the most number of profile ticks taken:

|  |  |
| --- | --- |
| i2c\_repeated\_read | 1754 |
| dmul | 1579 |
| i2c\_read\_setup | 888 |
| \_double\_epilogue | 429 |
| dadd | 305 |