Name: Emil Prisquilas Peter Student ID:200269166

ECE 561

Project 1: Speed Analysis and Optimization Report

Introduction:

This project is aimed at reducing the execution time for a given code. The code decodes a jpeg image stored in memory and displays the image on LCD screen. The decoded pixel data byte is cross checked with the correct byte stored in memory to ensure that function of the code is not changed with any optimization. Profiler samples the code at a sampling frequency of 1kHz.The total number of samples and the number of samples for each function is displayed on the LCD screen. The goal of the project is to obtain 435 samples with appropriate optimizations.

Execution Time Optimization:

Sample count for startup code:

	Function Name	Sample Count
Total Samples	**n/a**	1189
Function 1	LCD_24S_Write_Data	323
Function 2	LCD_Plot_Pixel	140
Function 3	pjpeg_load_from_memory	94
Function 4	LCD_24S_Write_Command	91
Function 5	Idctcols	67

Optimization 1:
Compiler optimization level was raised from level O1 to O3 and optimizing for time was enabled. Due to this the compiler performs more aggressive optimizations like automatic and aggressive inlining and loop unrolling and also the compiler optimizes the code for speed rather than size.
Time taken for optimization: 5 minutes
Sample counts before execution: 1189
Sample counts after execution: 1014
The performance improved by a speedup factor of **1.172**.
The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	1014
Function 1	LCD_24S_Write_Data	266
Function 2	decodeNextMCU	176
Function 3	LCD_Plot_Pixel	147
unction 4	LCD_24S_Write_Command	133
Function 5	pjpeg_load_from_memory	81

Optimization 2:
LCD_Plot_Pixel function call in the pjpeg_load_from_memory is replaced with LCD_Start_Rectangle and LCD_Write_Rectangle_Pixel function calls. LCD_Plot_Pixel function call sets column address and row address every time for every pixel, hence performing 12 LCD operations per pixel. Since the LCD is capable of accepting multiple data value after the column address and row address is set once, LCD_Start_Rectangle and LCD_Write_Rectangle_Pixel fucntions are used reducing the LCD operations to 10 + 2/pixel. Hence a great number of samples is expected to reduce with this optimization.
Time taken for optimization: 75 minutes
Sample counts before execution: 1014
Sample counts after execution: 577
The performance improved by a speedup factor of **1.757**.

The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	577
Function 1	decodeNextMCU	171
Function 2	huffDecode	73
Function 3	pjpeg_load_from_memory	67
Function 4	LCD_Write_Rectangle_Pixel	62
Function 5	LCD_24S_Write_Data	52

Optimization 3:

In this optimization 4 pixel information for each color was passed to the LCD_Write_Rectangle_Pixel function as parameters. The LCD_Write_Rectangle_Pixel function was modified to pack these 4 color information into 4 corresponding pairs of b1& b2. The function sent data to plot 4 pixels instead of one. This optimization was an attempt to parallelize the packing of 4 pixels into b1 & b2 to utilize the 32-bit width of registers. This optimization was aimed at reducing some overhead by sending out 4 pixel information in a single loop iteration. This optimization also cuts down the overhead involved in context switching for each pixel.

Time taken for optimization: 120 minutes

Sample counts before execution: 577

Sample counts after execution: 555

The performance improved by a speedup factor of **1.039**.

The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	555
Function 1	decodeNextMCU	159
Function 2	huffDecode	78
Function 3	LCD_Write_Rectangle_Pixel	70
Function 4	LCD_24S_Write_Data	65
Function 5	upsampleCr	45

Optimization 4:

This optimization replaces the use of PDOR with PSOR and PCOR in the GPIO_Write definition. This optimization speeds up the performance since it eliminates the overhead of writing 32-bit data to the PDOR register. Writing to PSOR and PCOR registers causes the modification of data in PDOR register only at certain specified bit positions, instead of at all positions.

Time taken for optimization: 120 minutes

Sample counts before execution: 555

Sample counts after execution: 499

The performance improved by a speedup factor of **1.112**.

The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	499
Function 1	decodeNextMCU	153
Function 2	huffDecode	84
Function 3	LCD_Write_Rectangle_Pixel	74
Function 4	upsampleCr	60
Function 5	upsampleCb	40

Optimization 5:

In this optimization the pointers pSrcR, pSrcG & pSrcB are passed as parameters to the LCD_Write_Rectangle_Pixel and whole operations for a MCU block is performed by the function. This eliminates the overhead involved in function switching for every 4 pixels and also eliminates unnecessary intermediate operations required for packing pixel information into b1 & b2 since b1 & b2 are directly formed using the pointers.

Time taken for optimization: 180 minutes

Sample counts before execution: 499

Sample counts after execution: 475

The performance improved by a speedup factor of **1.05**.

The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	475
Function 1	decodeNextMCU	163
Function 2	LCD_Write_Rectangle_Pixel	76
Function 3	huffDecode	71
Function 4	upsampleCr	51
Function 5	upsampleCb	50

Optimization 6:

In this optimization data corresponding to 4 pixels i.e. 4 bytes each were packed into b1 &b2. Hence this helped in reducing the number of operations required for packing 4 pixel data into b1 & b2 from 4 to 1, since earlier these operations were performed each time for every pixel.

Time taken for optimization: 240 minutes

Sample counts before execution: 475

Sample counts after execution: 453

The performance improved by a speedup factor of **1.048**.

The following is the count obtained for the top five functions:

	Function Name	Sample Count
Total Samples	**n/a**	453
Function 1	decodeNextMCU	144
Function 2	huffDecode	90
Function 3	upsampleCb	55
Function 4	LCD_Write_Rectangle_Pixel	55
Function 5	upsampleCr	40

Optimization 7:

In this optimization, the operations performed within LCD_24S_Write_Data function is written within LCD_Write_Rectangle_Pixel function and GPIO_SetBit(LCD_D_NC_POS) is called only once for a MCU block. This eliminates the overhead of calling GPIO_SetBit(LCD_D_NC_POS) every time for every 4 pixels, since before sending data to the LCD only one time it has to be communicated to the LCD that data is being sent.

Time taken for optimization: 60 minutes

Sample counts before execution: 453

Sample counts after execution: 431

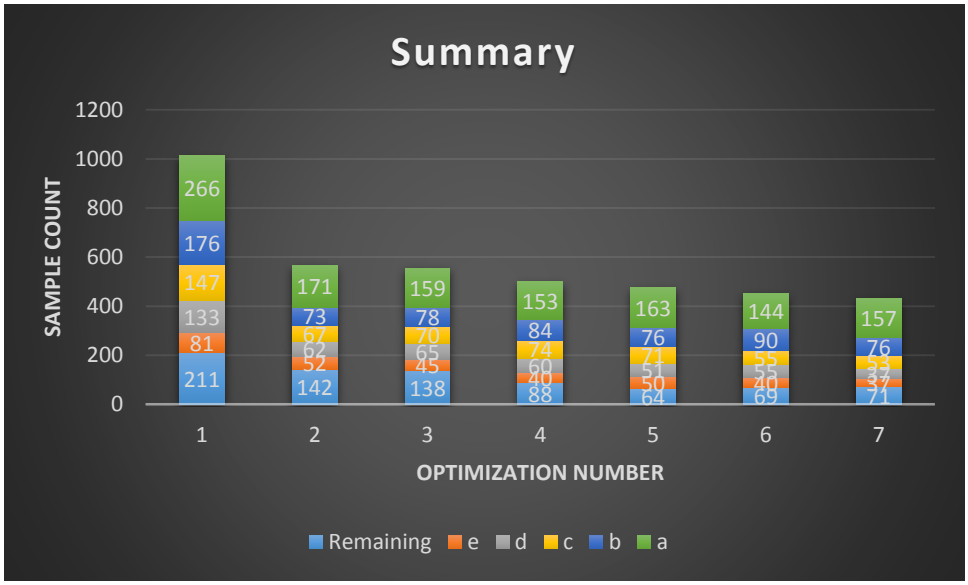
The performance improved by a speedup factor of **1.051**.

The following is the count obtained for the top five functions:

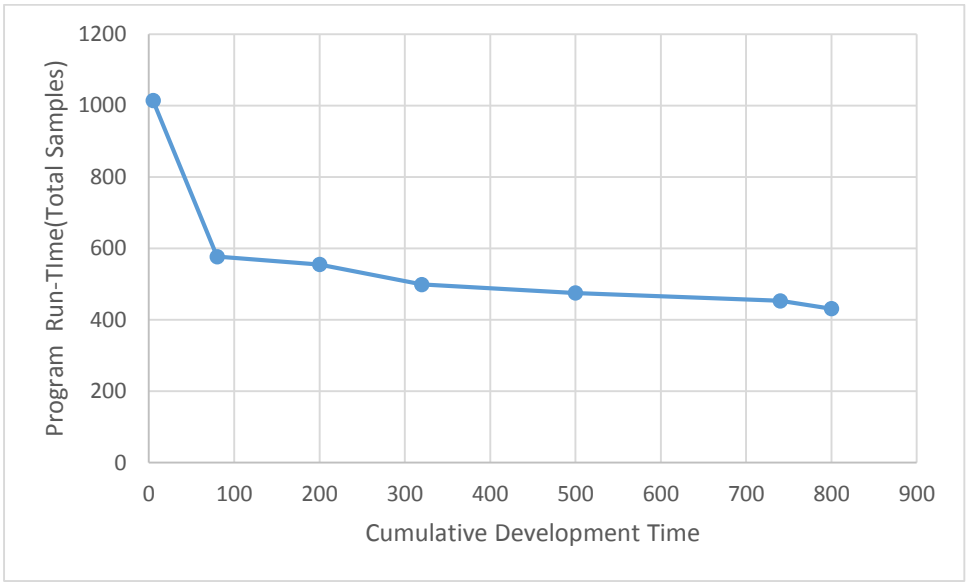
	Function Name	Sample Count
Total Samples	**n/a**	431
Function 1	decodeNextMCU	157
Function 2	huffDecode	76
Function 3	upsampleCb	53
Function 4	upsampleCr	37
Function 5	LCD_Write_Rectangle_Pixel	37

Analysis and Discussion of Results:

Summary stacked column chart:



Execution time v/s Development time:



Observations:

Optimization 2 of using LCD_Start_Rectangle, and LCD_Write_Rectangle_Pixel improved performance the most.
Optimization 1 of using level O3 compiler optimization offered the best improvement/development time.

Lessons Learnt:

- Understood the process of decoding a jpeg image.
- Understood procedure to be followed while optimizing a given code.
- Understood the use of __attribute__((noinline)) to disable the compiler from inlining functions.