Embedded Systems Design

Project 2

Mazarakis Periklis A.M.: 57595

Papadopoulos Aristeidis A.M.:57576

General:

In certain for loops, with nested switch cases, the start of the loop iterator was set to start from the next number and end in the second to last, as to not check the limits of the image. This optimization was made on the loops that calculate the convolution product. In the previous project, we considered the memory and the read/write times to be ideal. But with the removal of this consideration, we expect significant increase in clock cycles due to this time overhead.

```
void gauss_filter(){
  printf("Create Gaussian Image...\n");
  for (i = 0; i < N; i++)
  {
    switch (i) {
      default:{
          for (j = 0; j < M; j=j+3)
          { switch (j) {
            default:{ //convolve gauss filter in grayscale image,grayscale is the Y Channel
              gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
              + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
              + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];
              gauss_img[i][j+1] = current_y[i-1][j] * gauss[0][0] + current_y[i-1][j+1] * gauss[0][1] + current_y[i-1][j+2] * gauss[0][2]
              + current_y[i][j] * gauss[1][0] + current_y[i][j+1] * gauss[1][1] + current_y[i][j+2] * gauss[1][2]
              + current_y[i+1][j] * gauss[2][0] + current_y[i+1][j+1] * gauss[2][1]+ current_y[i+1][j+2] * gauss[2][2];
              gauss_img[i][j+2] = current_y[i-1][j+1] * gauss[0][0] + current_y[i-1][j+2] * gauss[0][1] + current_y[i-1][j+3] * gauss[0][2]
              + current_y[i][j+1] * gauss[1][0] + current_y[i][j+2] * gauss[1][1] + current_y[i][j+3] * gauss[1][2]
              + current_y[i+1][j+1] * gauss[2][0] + current_y[i+1][j+2] * gauss[2][1]+ current_y[i+1][j+3] * gauss[2][2];
            }
            case 0:continue;
            case M-1:continue;}
    case 0:continue;
    case N-1:continue;}
    }
}}}
```

Image 1 Code before change

```
void gauss_filter(){
  int gauss[3][3]= {{1, 2, 1},{2, 4, 2},{1, 2, 1}};
  printf("Create Gaussian Image...\n");
  for (i = 1; i < N-1; i++)
  {
          for (j = 0; j < M; j=j+3)
          { switch (j) {
            default:{ //convolve gauss filter in grayscale image,grayscale is the Y Channel
              gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
              + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
              + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];
              gauss_img[i][j+1] = current_y[i-1][j] * gauss[0][0] + current_y[i-1][j+1] * gauss[0][1] + current_y[i-1][j+2] * gauss[0][2]
              + current_y[i][j] * gauss[1][0] + current_y[i][j+1] * gauss[1][1] + current_y[i][j+2] * gauss[1][2]
              + current_y[i+1][j] * gauss[2][0] + current_y[i+1][j+1] * gauss[2][1]+ current_y[i+1][j+2] * gauss[2][2];
              gauss_img[i][j+2] = current_y[i-1][j+1] * gauss[0][0] + current_y[i-1][j+2] * gauss[0][1] + current_y[i-1][j+3] * gauss[0][2]
              + current_y[i][j+1] * gauss[1][0] + current_y[i][j+2] * gauss[1][1] + current_y[i][j+3] * gauss[1][2]
              + current_y[i+1][j+1] * gauss[2][0] + current_y[i+1][j+2] * gauss[2][1]+ current_y[i+1][j+3] * gauss[2][2];
            }
            case 0:continue;
            case M-1:continue;}
    }
}}
```

Image 2 Code after change

Memory hierarchy 1:

The memory hierarchy we defined consists of a ROM memory of 512KB, a DRAM RAM memory of 3.56MB and an SRAM memory that works like a cache memory of 1MB. The SRAM memory is on-chip, while the other two are off-chip. RAM memory size was set relatively large, due to the complex nature of the convolutions made in our project. Similarly, SRAM was set with large memory size, because the matrices that were more commonly accessed have large dimensions. So, the memory structure that was used can be seen in the following table (mapped at the file *memory.*map).

Initial memory address	Memory size (hex)	Memory name	Bus size (Byte)	Memory usage	Read time N/S	Write time N/S
0x0	0x0080000	ROM	4	R(=read)	150/100	150/100
0x0080000	0x0390C3C	RAM	4	RW(read,write)	50/25	50/25
0x0410C3C	0x0100000	SRAM	4	RW(read,write)	1/1	1/1

The times were used almost set arbitrarily, with the ROM times being the highest, since ROM memory is the slowest and off-chip, while RAM times are low, despite RAM memory being off-chip. Furthermore, SRAM is the fastest out of the three memories, due to the fact that SRAM memory is on chip. Finally, serial memory calls are the fastest, as the elements are nearly saved, so the time of serial accessing is lower than those of not serial accessing.

Furthermore, stack/heap structure is used, which is part of the RAM memory. It was modified in a way that it can contain the local variables and other elements, without overlapping with the other memory types. This happened with the change of its initial memory address (0x0041043C) and its final (0x00410C3C), i.e., the difference between heap and stack is 2KB approximately.

Respectively, the scatter file, determining the type of data saved in every memory type has the following format:

```
ROM 0x0 0x00080000
{
ROM 0x0 0x00080000
{
*.o ( +RO )
}
RAM 0x00080000 0x00390C3C
{
* ( ram )
* ( +ZI, +RW )
}
SRAM 0x00410C3C 0x00100000
```

```
{
*( sram )
}
}
```

Trials:

Initially, all our program's variables were placed in the DRAM memory, resulting in a huge increase of the clock cycles, since waiting cycles exist now, where the processor is idle or in waiting, as it waits for the data to be fetched from the memories. In our next test, we placed the loop iterators i,j in the SRAM memory, since they are used in every iteration. So, we expected a decrease in clock cycles, which indeed happened.

Moreover, by placing the matrices named *Ix,iy* and the variables *max,min* further clock cycle decrease was noticed. However by removing those matrices and by placing the *dI,theta* matrices in the SRAM memory, more clock cycle decrease was achieved. Finally, by storing the *max,min* variables in the SRAM, we noticed the optimal results, as it can be seen in the following table.

Test	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	Wait_States	Total
All variables in DRAM	350314447	492908467	392398885	74644413	91137539	2047332140	2605512977
i,j in SRAM	350314444	492908462	392398881	74644412	91137539	2039900279	2598081111
i,j,*Ix,iy* in SRAM	350314444	492908462	392398881	74644412	91137539	2038038791	2596219623
i,j,*Ix,iy,max, min* in SRAM	350314444	492908462	392398881	74644412	91137539	2037805211	2595986043
i,j,*dI,theta* in SRAM	350314444	492908462	392398881	74644412	91137539	2037892789	2596073621
i,j,*dI,theta, max,min* in SRAM	350314444	492908462	392398881	74644412	91137539	2037659209	2595840041

In the green-colored line the optimal results can be seen. Nevertheless, the use of SRAM memory with size 1MB is unusable, as the size is too large and thus, we had to reduce it, knowing that this kind of change would negatively impact the programs clock cycles, since the wait states would increase.

Bus size change:

A trial that was used is the change of the bus size. In our previous tests, the bus size was 4 Bytes (32bits), and now a smaller 2 Bytes (16bit) was used. We expect the clock cycles to double, as for a 4Byte transfer two clock cycles are needed instead of one.

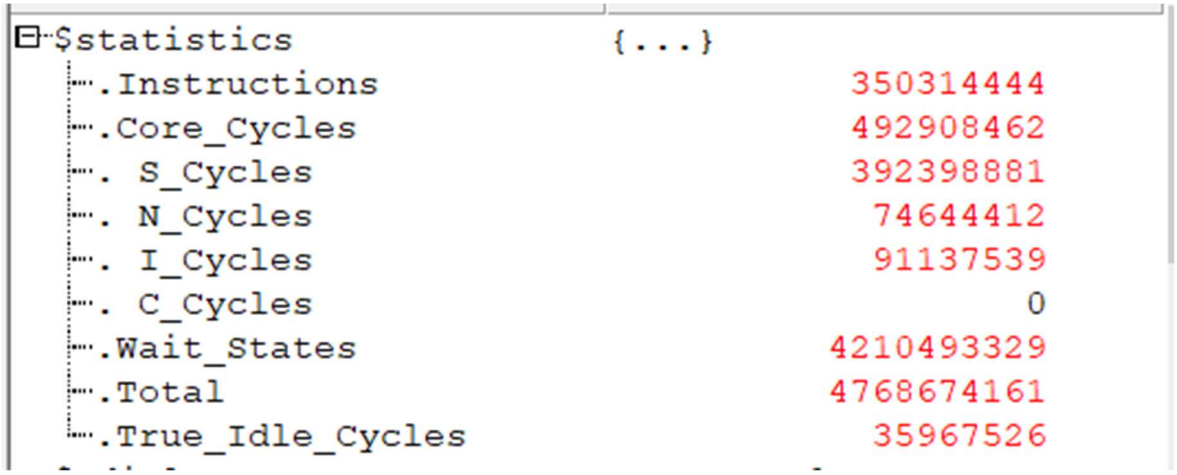

Image 3 Results from changing bus size

Indeed, the clock cycles were increased, as the wait states were doubled. Similarly, if the bus size is changed to 8 Bytes (64bits), the wait states will decrease in half, resulting in an overall clock cycle decrease. However, this case cannot be implemented.

Memory Hierarchy 2:

So, the new SRAM memory size is 64KB, with the remaining memory types being as they were. Trying different possible optimizations, we concluded that SRAM is too small to fit any of our data matrices, only the variables *i,j,max,min* with the results being available in the following table.

Test	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	Wait_States	Total
All the variables in DRAM	350314447	492908467	392398885	74644413	91137539	2047332140	2605512977
i,j in SRAM	350314444	492908462	392398881	74644412	91137539	2039900279	2598081111
i,j,*max,min* in SRAM	350080923	492207899	392165360	74410891	90904018	2036397405	2593877674
i,j in SRAM with 2byte bus	350314444	492908462	392398881	74644412	91137539	4213854934	4772035766
i,j,*max,min* in SRAM with 4byte bus	350080923	492207899	392165360	74410891	90904018	4207900060	4765380329

Optimal results can be seen in the green-colored line. Also, since only the variables i,j,*max,min* can fit in the SRAM memory, its size can be reduced to 12 Bytes, since any attempt to fit a matrix requires an extra 114KB (using the car image).

Loop Tiling Revisited:

As it is already known, loop tiling optimization works with a memory hierarchy. However, when we used our first memory hierarchy results were not what we expected at all, with the clock cycle increasing instead of decreasing.

Debugger Internals

Internal Variables | Statistics

Variable Name	Value
$statistics	{...}
.Instructions	356123697
.Core_Cycles	503239335
. S_Cycles	399054481
. N_Cycles	77079936
. I_Cycles	92769945
. C_Cycles	0
.Wait_States	2012913737
.Total	2581818099
.True_Idle_Cycles	36137109
$rdi_log	0

Image 4 Results after loop tiling

Loop tiling was implemented on the functions color & magn_theta_calc.

Conclusions:

Due to the problem's magnitude (3 convolutions, multiple if statements etc.) the use of several computational resources is needed. The use of RAM memory with size 3.5MB and SRAM 1MB size in the first memory hierarchy is only natural, even though their sizes might be considered big, especially for the SRAM memory, where actually creating this SRAM could be expensive. In any case, using a memory hierarchy increases the clock cycles due the increase of wait state cycles, since now accessing its memory requires more time than before. So, the ideal scenario would be the placement and storing of all our data in the SRAM on-chip memory, to minimize the transfer/wait times.

$statistics	{...}
.Instructions	350314447
.Core_Cycles	492908465
. S_Cycles	392398885
. N_Cycles	74644412
. I_Cycles	91137538
. C_Cycles	0
.Wait_States	0
.Total	558180835
.True_Idle_Cycles	35967526
$rdi_log	0
$target_fpu	1
$image_cache_enable	0
$clock	11163616

Image 5 Ideal memory

Where it can be seen that the cycles where the processor is idle have been minimized, essentially the results of the previous project.

The files used for the 1st memory hierarchy are:

- mymem1.map
- myscatter1.txt
- stack1.c
- optimized.c

The files used for the 2nd memory hierarchy are:

- mymem2.map
- myscatter2.txt
- stack2.c
- optimized2.c

The files used for the ideal memory hierarchy are:

- ideal.map
- ideal_scatter.txt
- ideal_stack.c
- ideal.c