Embedded Systems Design

Project 3η

Mazarakis Periklis A.M.: 57595

Papadopoulos Aristeidis A.M.:57576

Introduction:

The purpose of this project is the further optimization of our code, reusing our data by implementing buffers. This technique is very popular in cases where the same data is accessed frequently, with the target being the decrease of memory accesses and wait states, as well as the exploitation of the speed that cache memory has. However, there is a chance that the overhead of creating the buffers and adding elements to them could minimize any possible gain of their usage.

Memory hierarchy & row buffers creation:

The new memory hierarchy used has similar speeds to those of the previous project, with the difference being that the size of the SRAM memory is increased, to account for the storing of buffers, which are three in number.

Initial Memory Address	Memory Size	Memory Type	Bus Size	Mode type	Read Times (N/S)	Write Times (N/S)
0x0000000	512KB	ROM	4Byte	Read	150/100	150/100
0x0080000	3MB	RAM	4Byte	Read/Write	50/25	50/25
0x0410C3C	4KB	SRAM	4Byte	Read/Write	1/1	1/1

The buffers are set to be stored in the SRAM memory, as we need them to be instantly available with as fast access as possible.

```
#pragma arm section zidata="sram"
int i,j,k;
int rowbuffer1[M],rowbuffer2[M],rowbuffer3[M];
#pragma arm section
```

1st implementation (file rowbuffers1.c):

Based on the algorithm shown in our 3rd laboratory lesson, we tried to implement something similar, modified to our needs for this project. So, three buffers which are stored in SRAM memory are used, in our effort to speed up our program. In our tries to cut down the program's clock cycles, we had to minimize the loop unrolling optimization, to utilize more efficiently the buffers. In the following table the results can be seen, compared with the results of the previous project:

row buffers placement	Total Cycles
Gauss function	2.618.127.255
Convolution functions(Gauss + Sobel)	2.636.439.261
Previous Project Optimal	2.593.877.674

In the submitted file, row buffers were used in all our functions. In sum, the use of buffers as shown in our lesson ends up in more clock cycles than before.

```
void gauss_filter(){
  printf("Create Gaussian Image...\n");
  for (i = 1; i < N-1; i++)
  {
    if (i==1){
        for (k=0;k<M;k++){ //initialize buffers outside of main loop, with first 3 rows
            rowbuffer1[k] = current_y[0][k];
            rowbuffer2[k] = current_y[1][k];
            rowbuffer3[k] = current_y[2][k];
            }
        }
    else {
    for (k=0;k<M;k++){ // fill buffers according to lab exercise
        rowbuffer1[k] = rowbuffer2[k];
        rowbuffer2[k] = rowbuffer3[k];
        rowbuffer3[k] = current_y[i][k];
    }}
      for (j = 1; j < M-1; j=j+2){
          //convolve gauss filter in grayscale image,grayscale is the Y Channel
            gauss_img[i][j] = rowbuffer1[j-1] * gauss[0][0] + rowbuffer1[j] * gauss[0][1] + rowbuffer1[j+1] * gauss[0][2]
            + rowbuffer2[j-1] * gauss[1][0] + rowbuffer2[j] * gauss[1][1] + rowbuffer2[j+1] * gauss[1][2]
            + rowbuffer3[j-1] * gauss[2][0] + rowbuffer3[j] * gauss[2][1]+ rowbuffer3[j+1] * gauss[2][2];
            gauss_img[i][j+1] = rowbuffer1[j] * gauss[0][0] + rowbuffer1[j+1] * gauss[0][1] + rowbuffer1[j+2] * gauss[0][2]
            + rowbuffer2[j] * gauss[1][0] + rowbuffer2[j+1] * gauss[1][1] + rowbuffer2[j+2] * gauss[1][2]
            + rowbuffer3[j] * gauss[2][0] + rowbuffer3[j+1] * gauss[2][1]+ rowbuffer3[j+2] * gauss[2][2];
        }
    }
}
```

As it can be seen in the above example, the buffer initialization happens once, while the filling of those buffers happens 276 approximately. However, we found a method solving this problem, sliding the buffer's content each time, in the same spirit as the sliding filter in the convolution product.

2nd implementation (file rowbuffers2.c):

Searching a more effective implementation, with more variable assignments, but without using a loop, we reduced the clock cycles from our previous implementation. More

specifically, after initializing the three buffers outside of our two loops, their values are slided over in.

```
void gauss_filter(){
  for(k = 0;k<M;k++){
    rowbuffer1[k] = current_y[0][k];
    rowbuffer2[k] = current_y[1][k];
    rowbuffer3[k] = current_y[2][k];
  }
  printf("Create Gaussian Image...\n");
  for (i = 1; i < N-1; i++)
  {
      for (j = 1; j < M-1; j++){
            //convolve gauss filter in grayscale image,grayscale is the Y Channel
              gauss_img[i][j] = rowbuffer1[j-1] * gauss[0][0] + rowbuffer1[j] * gauss[0][1] + rowbuffer1[j+1] * gauss[0][2]
              + rowbuffer2[j-1] * gauss[1][0] + rowbuffer2[j] * gauss[1][1] + rowbuffer2[j+1] * gauss[1][2]
              + rowbuffer3[j-1] * gauss[2][0] + rowbuffer3[j] * gauss[2][1]+ rowbuffer3[j+1] * gauss[2][2];
            // metakulisi twn stoixeiwn twn rowbuffers
            rowbuffer1[j-1] = rowbuffer2[j-1];
            rowbuffer2[j-1] = rowbuffer3[j-1];
            rowbuffer3[j-1] = current_y[i+1][j-1];
        }
        rowbuffer1[j] = rowbuffer2[j];
        rowbuffer1[j+1] = rowbuffer2[j+1];
        rowbuffer2[j] = rowbuffer3[j];
        rowbuffer2[j+1] = rowbuffer3[j+1];
        rowbuffer3[j] = current_y[i+1][j];
        rowbuffer3[j+1] = current_y[i+1][j+1];
    }
}
```

Now, we can see that the elements change value in each loop, but with this method a third loop isn't used, and the changes are made in a more rational way.

Row buffer placement	Final Cycles
Gauss calculation function	2.618.711.872
Convolution Functions(Gauss + Sobel)	2.641.844.966
Previous project optimal	2.593.877.674

However, our initial target of reducing the total clock cycles from the previous project is not achieved. We deduced that this is happening for the same reason as it was for the previous implementation, i.e., the renewal of the buffers' values costs more than their usage.

Conclusions - Comments:

In sum, where we expected to see optimization in our program using buffers, especially since the convolution product is calculated, the result was the total opposite. In every implementation the total clock cycles were increased. We believe that this is because a lot of mathematical equations need to be used, such as the square root and atan. We tried using buffers in the functions where those two equations were used, but an increase in clock cycles was the results.