ESIEE

Computer Architecture

Optimisation des temps de calcul et processeur RISC Projet “Vision” sur carte SABRE i.MX_6, Linux OS et Raspberry Pi OS

Sahar Hosseini, Quoc-Trung Pham

October 2018

Objective:

The objective of this project is to experiment the optimization techniques in the context of an image processing embedded on the board SABER IMX6 (ARM_Cortex-A9, quad-core):

- algorithmic optimization (algorithm of lower complexity in the mathematical sense, adapted data structures)
- optimization for RISC processors
- optimization for efficient use of hardware resources: data access, cache management, multi-threading.

Tools:

Linux OS, Raspberry OS, board SABER IMX6, OpenCV library with C language

Introduction:

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

Implementation:

1. Implementing sobel filter

1.1 First version

We have computed the values of gradient pixel by pixel in direction of x and y with two functions, xGradient and yGradient. After reading the frame from camera and convert the BGR image to grayscale then applied median filter. By xGradient function we have computed the x component of the gradient vector at a given point in an image then returns gradient in the x direction and by yGradient function we have computed the y component of the gradient vector at a given point in an image returns gradient in the y direction.

```
int xGradient(Mat image, int x, int y){

      return image.at<uchar>(y - 1, x - 1) +

            2 * image.at<uchar>(y, x - 1) +

            image.at<uchar>(y + 1, x - 1) -

            image.at<uchar>(y - 1, x + 1) -

            2 * image.at<uchar>(y, x + 1) -

            image.at<uchar>(y + 1, x + 1);

}

int yGradient(Mat image, int x, int y){

      return image.at<uchar>(y - 1, x - 1) +

            2 * image.at<uchar>(y - 1, x) +

            image.at<uchar>(y - 1, x + 1) -

            image.at<uchar>(y + 1, x - 1) -

            2 * image.at<uchar>(y + 1, x) -

            image.at<uchar>(y + 1, x + 1);

}
```

we create the destination image with the same size of orginal image

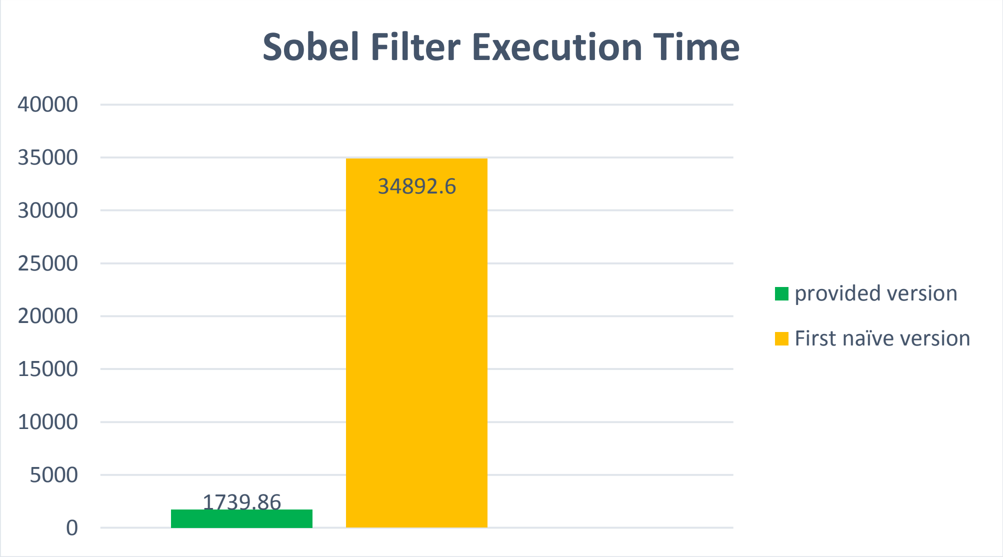
```
// -----------------REPLACE START--------------------------------

for (int y = 0; y < median.rows; y++)

      for (int x = 0; x < median.cols; x++)

            dst.at<uchar>(y, x) = 0.0;
```

we read image each row and each column compute gradient and them compute some of abs of them we do not need negative value after that we check this condition if sum of gradient x and gradient y is greater than 255 we set 255 in image(y,x) else we set the value of sum in image(y,x). If the sum is less than zero we set zero else we set sum in image(y,x).

```
for (int y = 1; y < median.rows - 1; y++){
  for (int x = 1; x < median.cols - 1; x++){
      gx = xGradient(median, x, y);
      gy = yGradient(median, x, y);
      sum = abs(gx) + abs(gy);
      sum = sum > 255 ? 255 : sum;
      sum = sum < 0 ? 0 : sum;
      dst.at<uchar>(y, x) = sum;
  }
}
// ----------------REPLACE END--------------------------------
```

1.2 Result

We have calculated the execution time for the first naïve version and provided version. The average of execution time for provided version is **1739.86 MicroSecond** and for first naïve version is **MicroSecond.**



Below table shows the configuration of system. The CPU is Intel with 3.30 GHz and the RAM size is 7926640 Kbytes.

Processor family	Frequency	RAM size	Cache memory organization
Intel(R) Xeon(R) CPU E3-1226 v3	3.30GHz	7926640 kiB	L1d cache:32K L1i cache:32K L2 cache: 256K L3 cache: 8192K

2.1 Optimize version

For optimization part first we remove some extra for and applied loop unrolling method in main for. We removed two functions xGradient and yGradient for computing Gy and Gy and compute it directly in the for loop.

```
// ----------------REPLACE START------------------------------
```

By comment this for because it is extra we can define the destination image by dst=frame.clone(); at the beginning of the program.

```
/*for (int y = 0; y < median.rows; y++)
      for (int x = 0; x < median.cols; x++)
            dst.at<uchar>(y, x) = 0.0;*/
```

here applied loop unrolling by increasing iterator of for to 2 and repeated the calculation part for computing gradient x and y.

```
for (int y = 1; y < width - 1; y=y+2){
     for (int x = 1; x < height - 1; x=x+2){
          gx = median.at<uchar>(y - 1, x - 1) +
            2 * median.at<uchar>(y, x - 1) +
            median.at<uchar>(y + 1, x - 1) -
            median.at<uchar>(y - 1, x + 1) -
            2 * median.at<uchar>(y, x + 1) -
            median.at<uchar>(y + 1, x + 1);
            gy = median.at<uchar>(y - 1, x - 1) +
            2 * median.at<uchar>(y - 1, x) +
            median.at<uchar>(y - 1, x + 1) -
            median.at<uchar>(y + 1, x - 1) -
            2 * median.at<uchar>(y + 1, x) -
            median.at<uchar>(y + 1, x + 1);
            sum = abs(gx) + abs(gy);
            sum = sum > 255 ? 255 : sum;
            sum = sum < 0 ? 0 : sum;

            dst.at<uchar>(y, x) = sum;
            gx = median.at<uchar>(y - 1, x+1 - 1) +
            2 * median.at<uchar>(y, x+1 - 1) +
            median.at<uchar>(y + 1, x+1 - 1) -
            median.at<uchar>(y - 1, x +1+ 1) -
            2 * median.at<uchar>(y, x+1 + 1) -
            median.at<uchar>(y + 1, x+1 + 1);
            gy = median.at<uchar>(y - 1, x+1 - 1) +
            2 * median.at<uchar>(y - 1, x+1) +
            median.at<uchar>(y - 1, x+1 + 1) -
            median.at<uchar>(y + 1, x +1- 1) -
            2 * median.at<uchar>(y + 1, x+1) -
            median.at<uchar>(y + 1, x + 1+1);
            sum = abs(gx) + abs(gy);
            sum = sum > 255 ? 255 : sum;
            sum = sum < 0 ? 0 : sum;
            dst.at<uchar>(y, x+1) = sum;

            gx = median.at<uchar>(y - 1+1, x - 1) +
            2 * median.at<uchar>(y+1, x - 1) +
            median.at<uchar>(y + 1+1, x - 1) -
            median.at<uchar>(y - 1+1, x + 1) -
            2 * median.at<uchar>(y+1, x + 1) -
            median.at<uchar>(y + 1+1, x + 1);
            gy = median.at<uchar>(y - 1+1, x - 1) +
            2 * median.at<uchar>(y - 1+1, x) +
            median.at<uchar>(y - 1+1, x + 1) -
            median.at<uchar>(y + 1+1, x - 1) -
            2 * median.at<uchar>(y + 1+1, x) -
            median.at<uchar>(y + 1+1, x + 1);
            sum = abs(gx) + abs(gy);
            sum = sum > 255 ? 255 : sum;
            sum = sum < 0 ? 0 : sum;
            dst.at<uchar>(y+1, x) = sum;
```

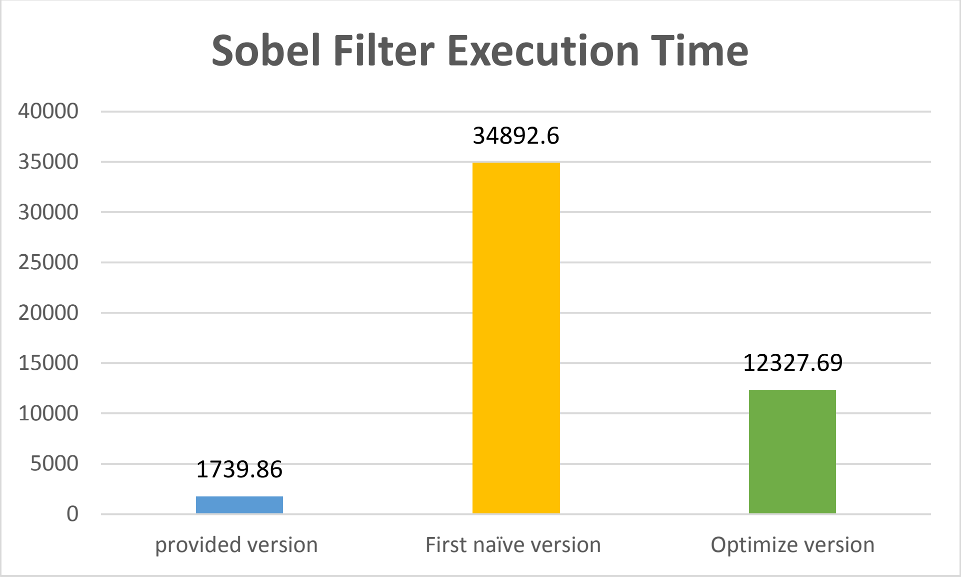
```
            gx = median.at<uchar>(y - 1+1, x - 1+1) +
            2 * median.at<uchar>(y+1, x - 1+1) +
            median.at<uchar>(y + 1+1, x - 1+1) -
            median.at<uchar>(y - 1+1, x + 1+1) -
            2 * median.at<uchar>(y+1, x + 1+1) -
            median.at<uchar>(y + 1+1, x + 1+1);
            gy = median.at<uchar>(y - 1+1, x - 1+1) +
            2 * median.at<uchar>(y - 1+1, x+1) +
            median.at<uchar>(y - 1+1, x + 1+1) -
            median.at<uchar>(y + 1+1, x - 1+1) -
            2 * median.at<uchar>(y + 1+1, x+1) -
            median.at<uchar>(y + 1+1, x + 1+1);
            sum = abs(gx) + abs(gy);
            sum = sum > 255 ? 255 : sum;
            sum = sum < 0 ? 0 : sum;
            dst.at<uchar>(y+1, x+1) = sum;
     }
}
// ----------------REPLACE END--------------------------------
```

2.2 Result

Below table shows the configuration of system. The CPU is Intel with 3.30 GHz and the RAM size is 7926640 Kbytes.

Processor family	Frequency	RAM size	Cache memory organization
Intel(R) Xeon(R) CPU E3-1226 v3	3.30GHz	7926640 kiB	L1d cache:32K L1i cache:32K L2 cache: 256K L3 cache: 8192K

After optimization the execution time decrease form the first version. The average of execution time for provided version is **1739.86 ms** and for first naïve version is **34892.6 ms** and for optimize version is **15797.46 ms.**



2. Comparison in different devices

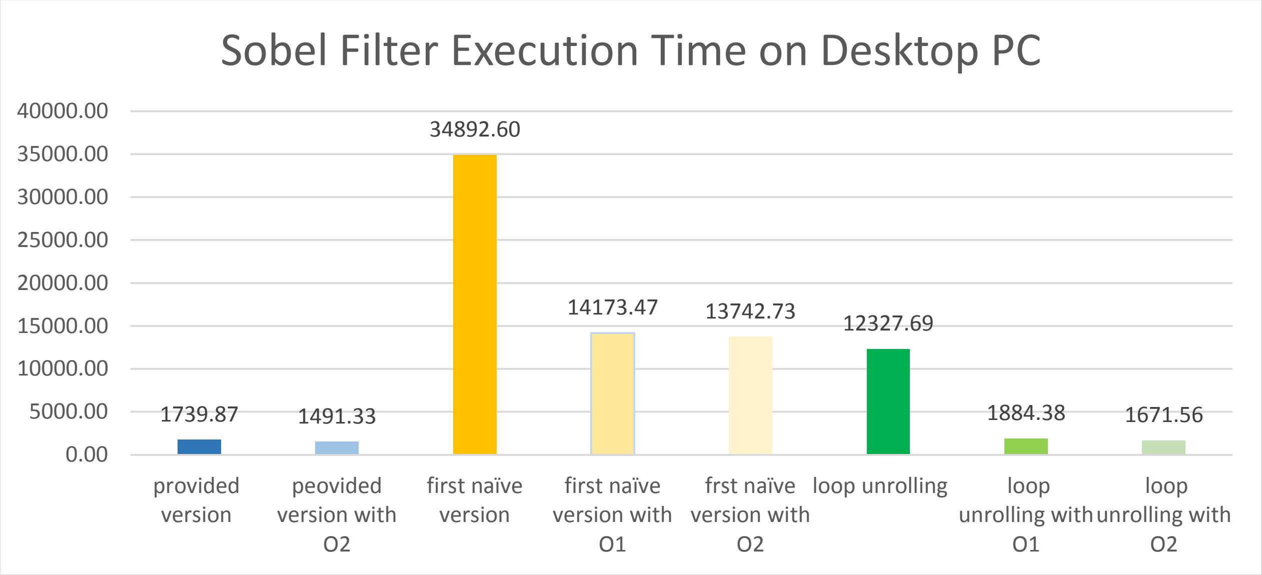
2.1 result of sobel filter execution time

Run the Sobel filter in these devices Linux, Sabre board and Raspberry. Also we run the program with compiler optimization and without compiler optimization. Below table show the configurations of these devices in Linux Terminal by adding O1 and O2 at the end of the compile command we change the compiling mode. The execution time for each devices and each version shows in the charts.

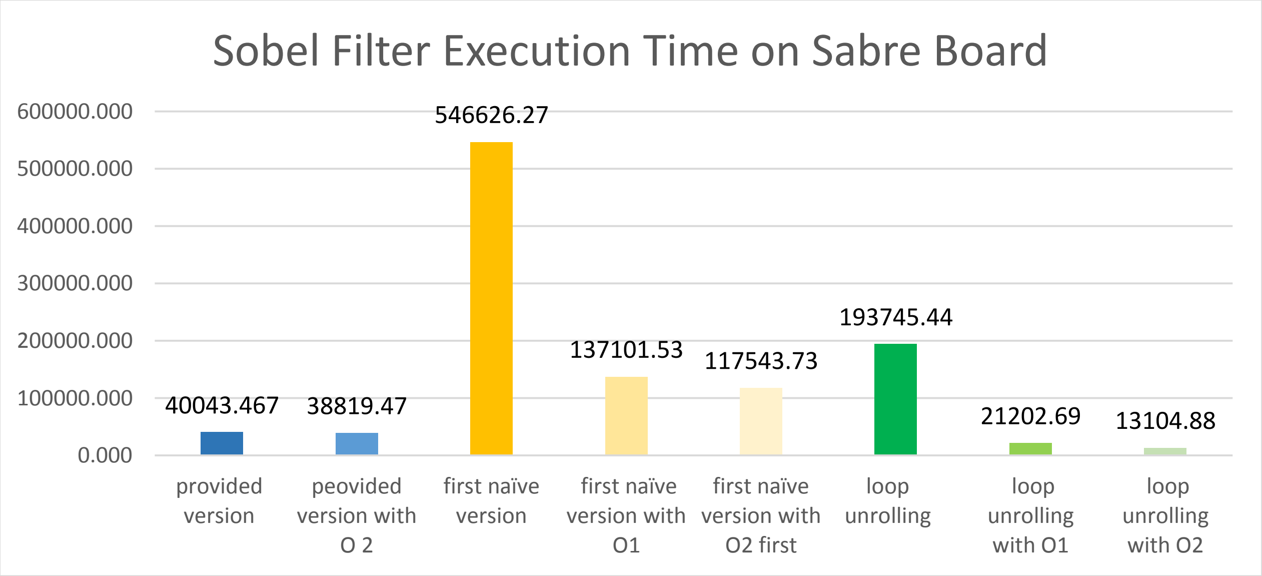
Device name	Processor family	Frequency	RAM size	Cache memory organization
Desktop PC	Intel(R) Xeon(R) CPU E3-1226 v3	3.30GHz	7926640 kiB	L1d cache:32K L1i cache:32K L2 cache: 256K L3 cache: 8192K
IMX.6 (SabreLite)	ARMv7 Processor rev 10 (v7l)	CPU max MHz: 996.0000 CPU min MHz: 396.0000	1028724 kiB	114760
Raspberry Pi	ARMv7 Processor rev 4 (v7l)	CPU max MHz: 1200.0000 CPU min MHz: 600.0000	883096 kiB	424845

Based on results, by using compiler options the execution time decrease especially with O2. And provided version in each devices has the best execution time. Using loop unrolling without compiler options has more execution time than first naïve version with compiler options in each devices.

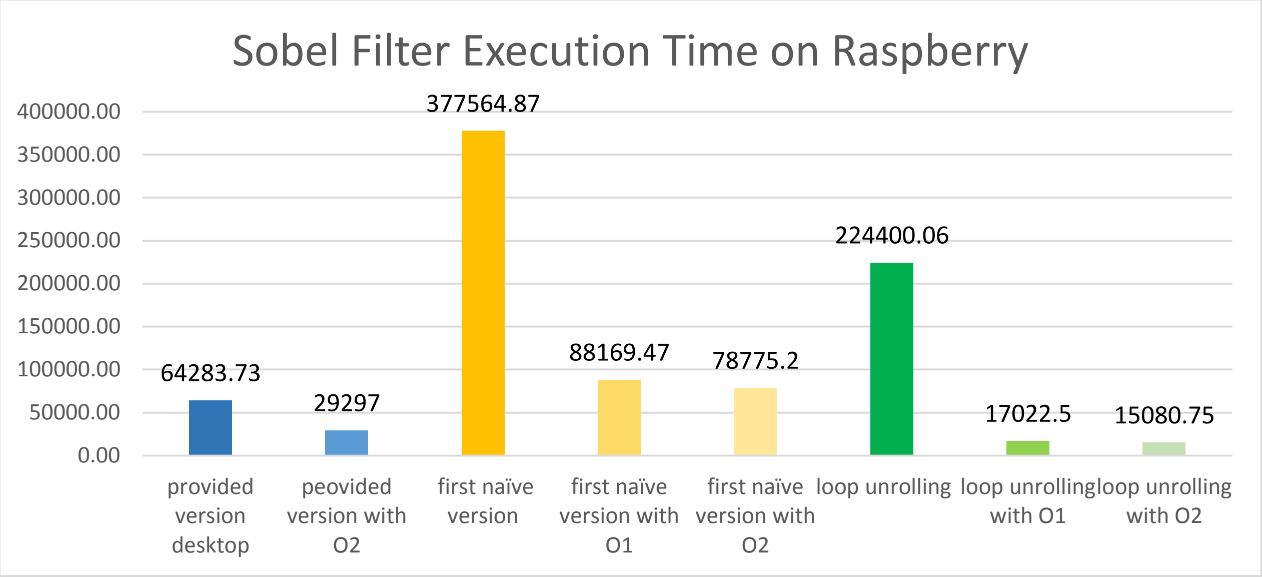
Below chart information show the execution time of each version of sobel filter with and without compiler optimization on Desktop PC. After optimization our result close to provided version with O2 compiler option.

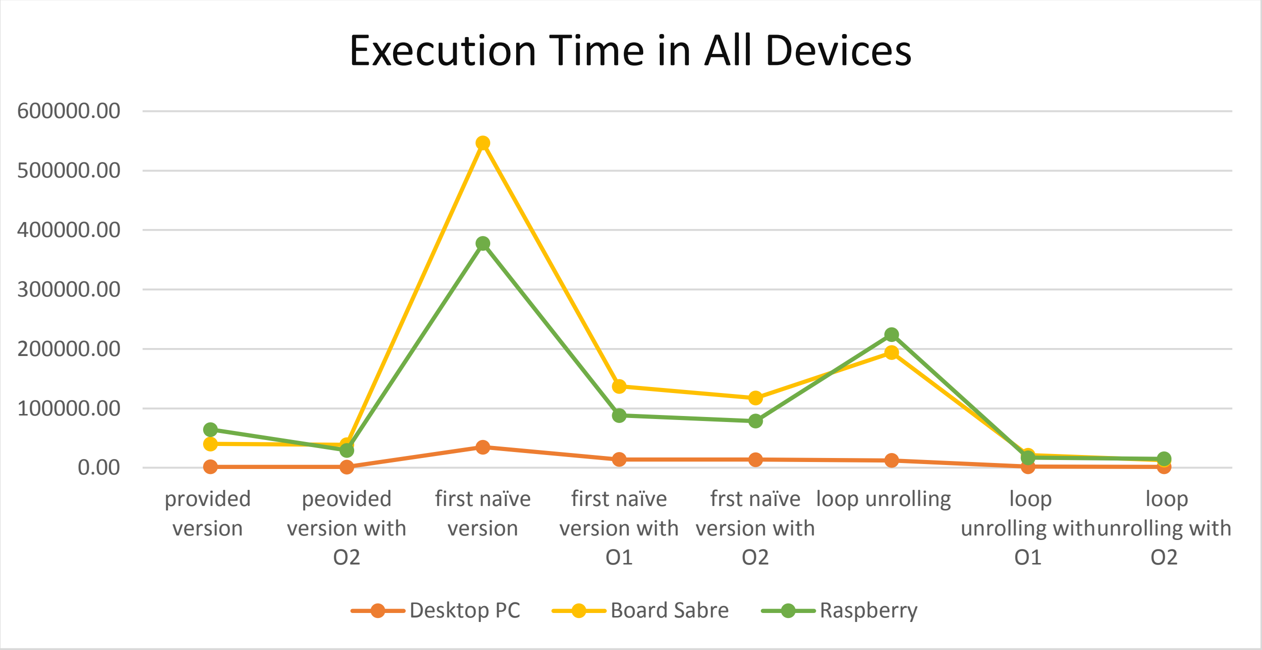


Below chart information show the execution time of each version of sobel filter with and without compiler optimization on IMX.6 (SabreLite).



Below chart information show the execution time of each version of sobel filter with and without compiler optimization on Raspberry Pi.





2.1 Conclusion

Provided version by opencv function of sobel filter in each devices has the best Execution time. But the value of execution time in Desktop has the lowest value because PC configuration in RAM and Frequency of processor is higher than other devices.

Execution time for optimize naïve version with O2 compiler in each devices has lowest execution time but in the PC has the best execution time and after that in RASPERRY Pi has lower value than sabre board. Also the execution time in provided version with O2 Compiler and Optimize version with O2 compiler a bit close in Sabre Board and Raspberry Pi but in PC NO.

First naïve version in each devices with O2 compiler has lower execution time that optimize version without compiler option in each devices.

So in result the execution time has directly depends on CPU and Memory (size & cache) configuration also using compiler options to compile the program is more effective than loop unrolling technics without compiler options based on the results that obtained.