

Cache Utilization Analysis of Edge Detection Kernels for Hardware Constrained Real-time Visual Sensor Devices

Usman Kaisser Moghal
Computer Sciences
FAST University
Islamabad, Pakistan
usmank.moghal@gmail.com

Abstract—With the rapid growth of Internet of Things (IOTs) devices and Wireless Sensor Network (WSN) nodes, there is a need for fast and efficient computing under heavy constraints. Certain sensor devices with cameras need to perform Edge Detection as a part of their functional process. This paper clearly defines which kernel filter for edge detection should be used in a cache constrained architecture to achieve optimal performance. Various images from the BSDS500 data-set are tested on using OpenCV's built in optimized libraries for convolving five of the most powerful and efficient kernels. Finally suggestions are given for how to extend this work.

Index Terms—Edge Detection Kernels, Cache Utilization Analysis, Wireless Sensor Devices, OpenCV, BSDS500

I. INTRODUCTION

As technology advances, more complicated tech gets squeezed into smaller roles, and user demands increase with this. Fast responsiveness is one major issue. Another being utility. With that in mind, the requirements of such modern devices to meet user requirements is very high.

This paper is aimed at one specific part of this problem. Among the plethora of ever advancing technology, sensor devices is one such area that sees some neglect. Very few researchers improve these devices by optimizing the hardware utility via software, and instead opt to either upgrade the hardware solely, or simply overhaul the whole software.

The paper is aimed at that particular lack in the software development community.

A. Cache

The hardware cache or processor cache is a memory that is inside the microprocessor itself and is a physical component of it. It is organized in different levels according to its proximity to the processor core (known as L1, L2 and L3 levels) and each of them is faster or slower depending on its proximity to the processor. The L1 and L2 levels usually have a very small capacity, just a few KB (much less than the capacities of RAM or the hard disk). In the case of cache memory, this is justified: the important thing is not the capacity, but the speed

B. Kernel

In image processing a Kernel is simply a 2-dimensional matrix of numbers which can range in dimensions. The use of Kernels - also known as convolution matrices or masks - is invaluable to image processing. Techniques such as blurring, edge detection, and sharpening all rely on kernels - small matrices of numbers - to be applied across an image in order to process the image as a whole.

C. Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a grayscale image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel.

D. Sensor Devices

A device that detects the changes in electrical or physical or other quantities and thereby produces an output as an acknowledgement of change in the quantity is called as a Sensor. Generally, this sensor output will be in the form of electrical or optical signal. The most frequently used different types of sensors are classified based on the quantities such as Electric current or Potential or Magnetic or Radio sensors, Humidity sensor, Fluid velocity or Flow sensors, Pressure sensors, Thermal or Heat or Temperature sensors, Proximity sensors, Optical sensors, Position sensors, Chemical sensor, Environment sensor, Magnetic switch sensor, etc.

II. RELATED WORK

Currently no work exists that showcase the hardware performance differences between different edge detecting kernels. Hence, the motivation for this work. However, some work has

been done with regards to CPU analysis for image processing, as found in [1], which details certain details that should be kept in mind when designing a CPU architecture for a specific image processing task.

Both [2] and [3] discuss those hardware optimization for real-time critical devices such as those used in IOT's or WSN's. [2] is more relevant for this paper as it specifically focuses on cache optimization for embedded systems, while [3] focuses on OpenCV.

And for general hardware efficiency optimization with respect to image processing, [4] and [5] are very relevant.

III. EXPERIMENTAL DETAILS

The experimental setup is simple yet thorough. The algorithm 1 shows an abstract view of the experimental setup.

Algorithm 1: Algorithm for recording performance counters

Input : Image data-set

Output: Performance record

Select 100 random images from BSDS500

Generate 4 size variant sets per image

Initialize kernels

foreach *set in sets* **do**

foreach *image in set* **do**

foreach *kernel in kernels* **do**

 convolve image using kernel

 record performance

end

end

end

The table I lists the details of the system used to test the experiment.

TABLE I
EXPERIMENTAL SYSTEM DETAILS

System Details	
<i>Model</i>	i7-4600U
<i>Cores</i>	4
<i>Threads</i>	4
<i>min MHz</i>	800
<i>max MHz</i>	3300
<i>Base Speed</i>	2.10
<i>Boost Speed</i>	3.30
<i>L1-I Cache</i>	32 KB 8-way associative
<i>L1-D Cache</i>	32 KB 8-way associative
<i>L2 Cache</i>	256 KB 8-way associative
<i>L3 Cache</i>	4 MB 16-way associative (shared)
<i>RAM</i>	8 GB

The following kernels are used for this experiment:

- 1) Canny, shown in figure 1. It is a 3x3 kernel, and has two masks, one for x-axis and one for y-axis.
- 2) Laplacian, shown in figure 2. It is a 3x3 kernel, and has only one mask.

- 3) Prewitt, shown in figure 3. It is a 3x3 kernel, and has two masks, one for x-axis and one for y-axis.
- 4) Robert, shown in figure 4. It is a 2x2 kernel, and has two masks, one for x-axis and one for y-axis.
- 5) Sobel, shown in figure 5. It is a 3x3 kernel, and has two masks, one for x-axis and one for y-axis.

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Fig. 1. Canny Kernel

1	1	1
1	-8	1
1	1	1

Fig. 2. Laplacian Kernel

0	1	1
-1	0	1
-1	-1	0

-1	-1	0
-1	0	1
0	1	1

Fig. 3. Prewitt Kernel

-1	0
0	1

0	-1
1	0

Fig. 4. Robert Kernel

For the sake of consistency, the built in *filter2D* function of the OpenCV library is used to convolve the selected kernels. Moreover, the standardized segmentation data set BSDS500 from Berkeley is selected for testing, as segmentation involves edge detection.

100 landscape images are chosen at random. To determine what the image input will have on the cache utilization, copies of each image is generated, such that a set of a total of 4

-1	0	1
-1	0	1
-1	0	1

1	1	1
0	0	0
-1	-1	-1

Fig. 5. Sobel Kernel

images now exist for each image at intervals of 25%. This is shown in the table II.

TABLE II
IMAGE SIZE VARIANTS

Image Dimension(px)	Size(%)
120x80	25
241x161	50
361x241	75
481x321	100

All image sets were convolved by each kernel a minimum of 50 times per size, and the average was calculated for each set. The final results have been recorded and displayed in table III.

For hardware analysis, *PERF* was used to record the following performance counters:

- Instructions
- Cache References
- Cache Misses
- Branches
- Branch Misses
- Time Taken

IV. RESULTS AND DISCUSSION

Table III details the results obtained from the aforementioned experimentation. These results are critically analysed based on the following perspectives:

A. Time

As visible from figure 6, all kernels displayed a consistent trend; a near linear increase in time taken as the image size is increased. However, it is important to note that the *laplacian* and *robert* kernel take convolve faster than the others. This is due to the decrease in computations required. *Laplacian* only has one mask while *robert* has a smaller size. Moreover, figure 6 shows that for smaller data sizes, the difference between kernels in terms of computation time is near insignificant.

B. Cache Miss Rate

Figure 7 shows cache miss rate for each kernel. It is evident that increasing the size also increases the cache miss ratio. However, this growth is expected to plateau at some point. The figure 7 also shows that almost all kernels except for *laplacian* result in a near consistent cache miss rate. This is a result of having one less mask to convolve.

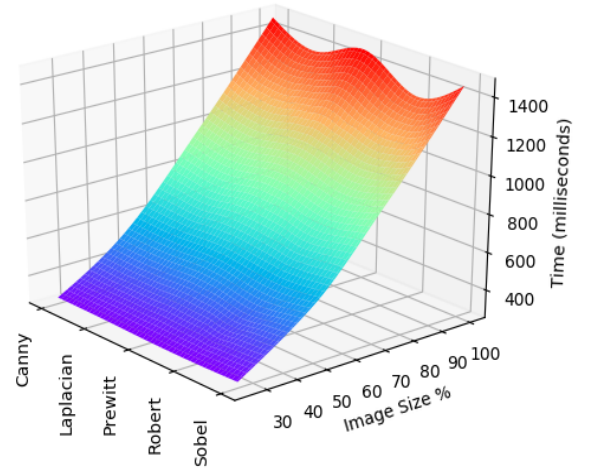


Fig. 6. Time

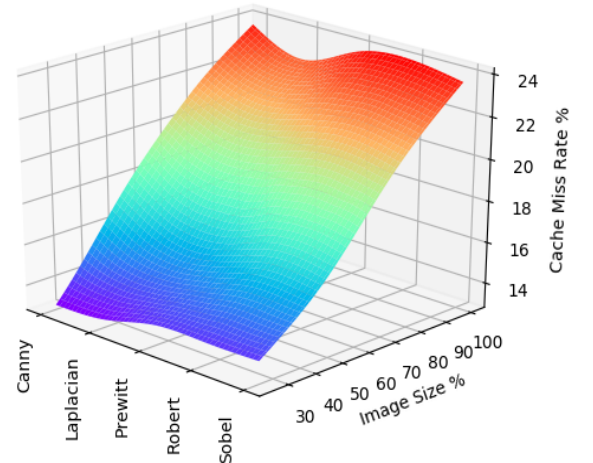


Fig. 7. Cache Miss Rate

C. Branch Miss Rate

Figure 8 shows that the *laplacian* kernel produces the largest branch miss rate for all image sizes, whereas the *robert* kernel produces the least branch miss rate for all image sizes. This, once again, co-relates to the *laplacian* kernel having only one mask, while the *robert* kernel having a smaller size.

However, the figure 8 shows some conflicting results. The branch miss rate was expected to grow with the increase in image size, however the figure clearly shows that after an increase of 75%, the miss rate goes down. The reason for this is currently yet to be explored, as it requires further testing.

TABLE III
RECORDED PERFORMANCE FOR CONVOLVING KERNELS

Kernel	Performance Counters						
	Image Size	Instructions	Cache References	Cache Misses	Branches	Branch Misses	Time (milliseconds)
Canny	25	1476328820	10755064	1401513	219550682	4092985	286.85461
	50	3497004297	11817497	2061381	403489116	8073209	560.511336
	75	6641507044	15192683	3206679	684635626	13943845	998.95188
	100	10118515979	19583894	4638590	990220714	18537080	1469.390907
Laplacian	25	1453045849	10082119	1328851	218032940	4094831	284.739076
	50	3393768672	11992468	1923643	396290024	8035786	552.253034
	75	6419294904	14130800	2855497	669467838	13859871	951.847791
	100	9631145652	17318492	3922104	950931895	17975786	1370.92965
Prewitt	25	1471145540	10238237	1414331	218880502	4059322	278.824865
	50	3472915997	12410463	2000203	400370912	7910817	561.91639
	75	6580015576	15264872	3143259	676506842	13552012	986.745612
	100	10017853954	19872861	4708716	976588576	17902489	1462.879475
Robert	25	1411688763	10186614	1412363	215196818	3949785	278.24447
	50	3229620060	11704914	1979525	385104860	7454521	527.362991
	75	6022363419	15286754	3218022	640686115	12482205	924.029387
	100	8986874473	19215944	4618578	907904788	15873344	1321.015443
Sobel	25	1471005933	10386831	1443804	218841047	4054434	285.160468
	50	3472032036	12083417	2017322	400158598	7891290	555.624086
	75	6576153778	15339194	3163445	675787470	13494423	980.779619
	100	10013883411	19521775	4615319	975736123	17797800	1452.379601

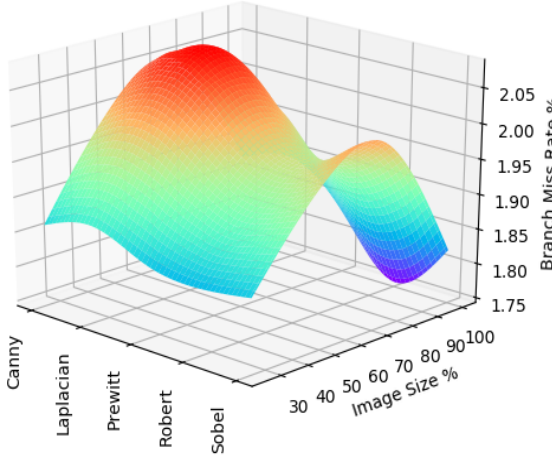


Fig. 8. Branch Miss Rate

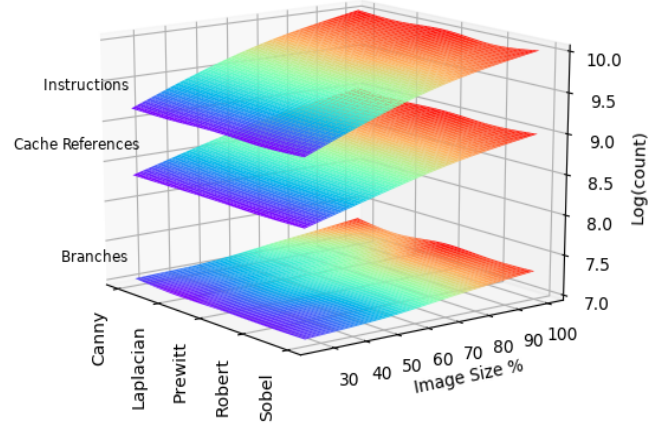


Fig. 9. Instruction/Cache references/Branches Count

However, as the miss rate fluxes between 1.75% and 2.05%, the difference can be seen as minimal.

Also important to note, that the remaining three kernels have a near identical branch miss rate.

D. Instruction Count

The figure 9 shows the count of instructions, cache references, and branches of the respective kernels across different image sizes. The general trend of increase in instruction count with increase in image size is maintained. However, as the image size increases, the count increases drastically. However,

the branch and cache reference count is barely changing despite working on a larger image.

Moreover, changing the kernel does not have any noticeable impact on cache reference or branch count.

E. T-test

To confirm a significant change in performance based on selection of kernel, a t-test was performed with a confidence of 95%. The aggregate results of the *canny*, *prewitt* and *sobel* filters is used as a baseline, to measure the difference with the *laplacian* and *robert* filters. The results can be seen in table IV.

TABLE IV
T-TEST

Kernel	Performance Counters	P-Value	Result
Laplacian	Cache Miss	0.000	Rejected
	Cache Miss	0.000	Rejected
	Branch Miss	0.000	Rejected
	Time Taken	0.012	Rejected
Robert	Cache Miss	0.048	Rejected
	Cache Miss	0.000	Rejected
	Branch Miss	0.000	Rejected
	Time Taken	0.000	Rejected

According to the t-test results, there is no significant difference between the means of the recorded values, hence the null hypothesis is rejected.

Despite seeing a noticeable difference in results, the t-test shows a different story. This needs to be further tested.

V. CONCLUSION

After experimentation and critical analysis of the produced results, the effects of kernel types on cache utilization become clear. Decreasing the number of masks will decrease the cache miss rate significantly, while having a minor increase in branch miss rate. Whereas decreasing the masks of a kernel does not effect miss rate at all, while marginally decreasing the branch miss rate. However doing either of both will decrease the processing time.

Considering these results, it becomes evident that to make an ideal edge detection software for an extremely hardware constrained device, the *laplacian* kernel should be used. Using the *laplacian* kernel will both decrease cache misses, while speeding up the process. Also, from other papers, we find that the *laplacian* kernel also produces a better result (considering PSNR and MS-SSIM).

It is also confirmed that if kernel size and mask count is kept the same, changing the values within the kernel does not have any significant change any of the counters.

The gains from using *laplacian* or *robert*, as compared to the other kernels are minimal, according to all tests. However, a minor gain in performance can lead to large gains when working on a larger problem. More testing is required for confirmation.

For future work, super resolution images and variable bit-rate images should be tested on, to also narrow down the optimal kernel based on the image being generated by the sensor device. Also, there are many other CPU hardware performance counters yet to be analysed, such as pre-fetch size, cache size, set associativity, etc. Moreover, additional testing needs to be performed to confirm the t-test results.

REFERENCES

- [1] Bruno Cernuschi-Frias, José Luis Hamkalo, Jonás D Pfefferman, et al. "Analysis of cache memory strategies for some image processing applications". In: *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*. Vol. 3. IEEE. 2001, pp. 334–337.
- [2] Abu Asaduzzaman. *Cache optimization for real-time embedded systems*. Florida Atlantic University, 2009.
- [3] Kari Pulli, Anatoly Baksheev, Kirill Korniyakov, et al. "Real-time computer vision with OpenCV". In: *Communications of the ACM* 55.6 (2012), pp. 61–69.
- [4] David Watson and Ali Ahmadiania. "Memory customisations for image processing applications targeting MP-SoCs". In: *Integration, the VLSI Journal* 51 (2015), pp. 72–80.
- [5] Antoine Colin and Stefan M Petters. "Experimental evaluation of code properties for wcet analysis". In: *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. IEEE. 2003, pp. 190–199.