Parallelizing Sobel Edge Detection

Michael Shlega
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
michaelshlega@cmail.carleton.ca

December 24, 2022

1 Introduction

Parallel computing has made great breakthrough's in the acceleration of computer vision algorithms. Edge detection algorithms are a foundational unit of any computer vision algorithm and research has been actively exploring ways of speeding up edge detection algorithms such as the Sobel operator. Despite these attempts, research has primarily focused on exploring the SIMD approach to improve speedup. We attempt to build on previous work and approach the problem by comparing the three parallel taxonomical approaches. We aim to understand which taxonomy provides the fastest speedup, which taxonomy provides the easiest implementation, and what the explanation for the differences in results is.

2 Literature Review

2.1 Edge Detection

Edge detection is a collection of mathematical methods which attempt to take in a digital image and identify the edges existing within it. It is an active area of research as the concepts used here are the foundation for many other fields - especially computer vision.

On a basic level, edge detection attempts to look for significant local change in image intensity. Discontinuities are either step discontinuities, or line discontinuities. The former is when we change rapidly from one value to another when going other a discontinuity. The latter involves the intensity of the image to change rapidly to a new value at some point, but then further down return to the same value [18, 19]. Unfortunately, real world images are often not as simple. Step edges turn into ramp edges and line edges become roof edges, with the changes of value occurring over a finite distance rather than abruptly [18]. Figure 1 demonstrates these differences.

Edge detection algorithms attempt to detect significant local change in the image, while not letting the noise of the image get in the way. They quantify the image as an array of image intensity based on a continuous function [4, 34, 31, 30]. Commonly, a step edge

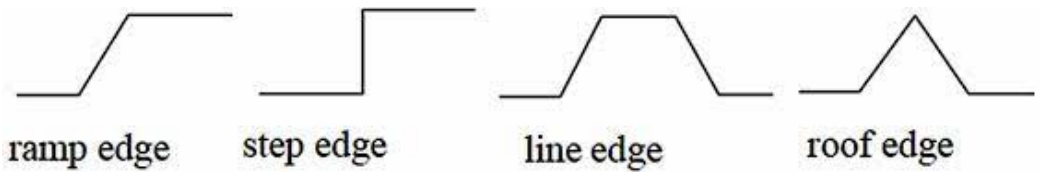

Figure 1: Examples of the four edge types

would be associated with a local peak in the first derivative of the graph mapping values to location [18]. However, this results in too many edge points. The solution is to find points that have local maxima in gradient values and consider them edge points. The two main approaches here are the Laplacian and second directional derivative. We will now explore the main operators used in edge detection, both the first order and second order.

2.1.1 Roberts Cross Operator

One of the first edge detection operators proposed, the Roberts cross operator utilises discrete differentiation to approximate the gradient. It first applies the Gx and Gy mask, which are 2x2, to the entire image from where it proceeds to calculate the gradient magnitude [31].

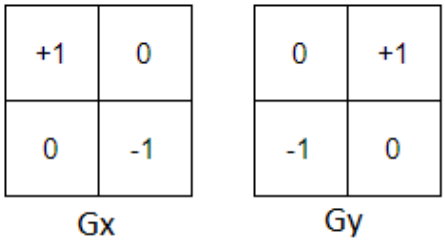

Figure 2: Roberts Operator Kernel

2.1.2 Sobel Edge Detection

This was the next main edge detection algorithm to come out, and one that is still prominently in use today. The Sobel operator uses a 3x3 image mask to convolute the image, using horizontal and vertical anchoring. The masks allow approximation in both the y and x direction, from where it can more accurately estimate the gradient[34].

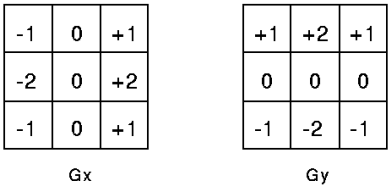

Figure 3: Sobel Operator Kernel

2.1.3 Prewitt Edge Detection

The Prewitt algorithm utilises a similar approach to the Sobel, however, it provided a slight computation speed improvement and also involves different kernels [30]. Note how the masks used for convolution do not place emphasis on the middle pixels as the Sobel kernels do.

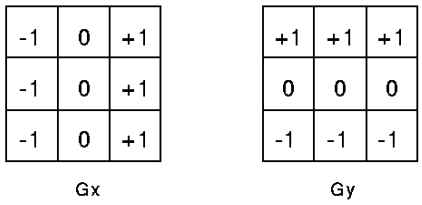

Figure 4: Prewitt Operator Kernel

2.1.4 Canny Edge Detection

The algorithm proposed by Canny et al. is a computationally effective algorithm that declares computational requirements for edge detection and proposes a highly effective circular operator meant to work on any scale [4].

2.1.5 Laplacian Edge Detection

Laplacian edge detection involves the estimation of the second derivative. The first step involves the use of a Gaussian filter to blur the image, making it smoother to work with. From there masks are used to create the resulting edge images (with various kernels being able to be used) but the kernels are meant to calculate the second order derivatives. One thing to note is that since we have one kernel with the Laplacian, it computers the result much faster. However, because it uses second derivatives, it is extremely sensitive to noise [18].

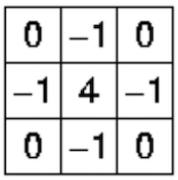

Figure 5: Laplacian Operator Kernel

2.2 Overall Edge Detection

The above mentioned operators are the ones most commonly used in current industry and academia[39]. The authors would like to mention other notable papers that have not made it into a section of their own, yet these papers have also provided effective algorithms and filter approaches yielding quality results [3, 14, 28, 15, 19].

2.3 Edge Detection Improvement

2.3.1 Algorithmic

One attempted approach in literature is to improve the problem of edge detection utilising improved algorithms. Dollar et al. attempt to obtain edge detection improvement through the use of effective algorithmic structure. Their approach is to implement a structured decision forest, which improves runtime, but at the cost of having a lower accuracy[9]. Other approaches have also been taken in academia, with the majority having a pattern of improving the runtime of the algorithms, but at the cost of the quality of output [20, 38, 23, 36, 13, 1] Some of these papers directly attempt to alter the main existing algorithms, such as the Canny or Sobel, while others attempt to alter the way the operators are utilised. Another algorithmic improvement carried out by Rosenfeld and Thurston [32] introduced smoothing operations in order to reduce the noise of the image prior to differentiation, thus improving the edge quality. Another approach carried out by Hueckel [16, 17] involved fitting each image pixel with a step edge in a circular window. If the fit was accurate, it was safe to assume that an edge exists with the same parameters as the fitted model. This is called parametric fitting and has been attempted by other researchers as well, with varying success [7, 25, 24, 2]. Another approach taken is by changing the implementation side [33, 35] .

2.3.2 Parallelism

Parallelism is an alternate method that utilises parallel code architecture to enable the speed up of the edge processing algorithms. There have been multiple approaches taken to implementing parallel edge detection, but they can be split up into two methods: (1) Pipeline parallelism and (2) Data parallelism [10]. Pipeline parallelism is possible because image processing requires a sequence of tasks, which can then be run concurrently on different processors. This is an example of MIMD classification parallelism. This has two distinct disadvantages though - the first is that the rate of throughput is dictated by the slowest task, with the second being that the software becomes non saleable as it is difficult to decompose into arbitrary amounts of concurrent tasks[10]. Certain papers have attempted this and shown limited speedup when compared to SIMD [33]. Data parallelism is the alternate option, and is more general purpose. It allows to partition the image into sections which can be assigned to different processors. The processors carry out the same task on each of the segments and then one processor builds the final result. This is an example of SIMD classification. Previous literature has shown multiple examples, with varying success [36, 37, 6]. This paper will build off of previous work and attempt to recreate previous experiments and analyze multiple operators based on their carryover to parallel code.

Let us here take a moment to expand on the theory behind various parallelism approaches seen in literature. The basis of all parallelism stems from Flynn's taxonomy [11]. There are four main types of parallelism.

1. **SISD**: Single instruction, single data - this is the most common architecture that has a single uni-core processor operate on the one piece of data. Such an approach is typically used in normal program running

2. **SIMD**: Single instruction, multiple data - architecture where allows processors to run the same instructions on different pieces of data. Most commonly implemented by GPUs.

3. **MISD**: Multiple instruction, single data - architecture where many instructions are run on the same data. This architecture is not commonly used, but has potential applications such as flight control computers.

4. **MIMD**: Multiple instruction, multiple data - architecture that allows processors to function on various pieces of data independently. Such architecture can be commonly found in distributed systems.

In relation to our study, previous work has primarily worked with MISD based parallelism implementation approaches. When the CUDA based GPU programming language came out - there was a surge of papers exploring the GPU edge detection approach [21, 22, **?**, 22, 29]. While previous research has demonstrated that MISD approaches have the highest efficiency of edge detection, we aim to expand on previous work by offering insight into other taxonomical approaches and comparing them.

3 Research Question

Our goal was to explore the various ways of re implementing the Sobel edge detection algorithm in parallel. We compare the various taxonomic implementations across three different variables: (1) efficiency of parallelism, (2) ease of implementation, and (3) quality of result.

4 Methodology

To answer our research question we took the Sobel algorithm and implemented a serial version in python through the use of openCV. Our serial version was used as a benchmark to compare the following parallel versions to.

To begin, let us explore how the implementation serial version of the Sobel operator works. We start with the following image:

Figure 6: Original image to be transformed

4.1 Serial Implementation

Our serial filter approaches the implementation by first applying Gaussian smoothing to the image to support the application of the filter later on. Research has demonstrated that smoothing leads to a better edge detection result [12]. We obtain the image shown in figure 7.

Figure 7: Image with Gaussian smoothing applied

After carrying out the smoothing of the image, we carry out the Sobel filter convolution. We calculate the image intensity at each pixel within the image by applying the Gx and Gy kernels mentioned previously 3. Our final result is shown in figure 8.

Figure 8: Edges of the original image

4.2 SIMD Implementation

The SIMD approach involves splitting up the image into multiple blocks. We take the amount of cores available to us and iterate over the image, splitting it into even chunks that we then allocate to each processor. These processors then seperately carry out the same convolution on these image chunks. From there, we reassemble the chunks to obtain out final image. Figure 9 demonstrates how the splitting of the image works - a grid is shown

on the image where the image was split with each chunk being assigned to a processor - which then carries out convolution on each pixel in the chunk.

.

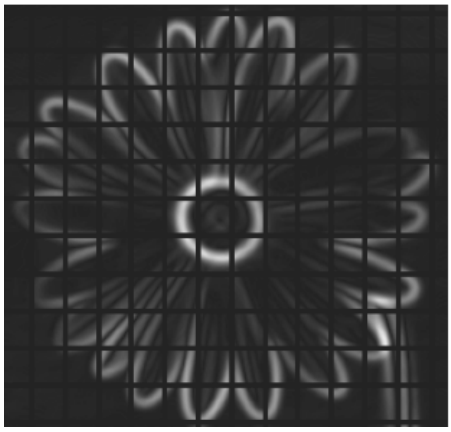

Figure 9: SIMD Image Splitting

4.3 MISD Implementation

The MISD approach involves approaching the problem from a parallel perspective and splitting up the kernel applications into two functions that can be run in parallel. In the serial implementation we have two for loops which are responsible for applying the convolution to each individual pixel. However, that for loop carries out two seperate operations - it first applies the Gx filter and then the Gy filter on the same pixel, from which we can then obtain the convolution. To optimize this, we utilise a SIMD approach and split up the filters to operate on the image in parallel. Once the parallel processes are finished we combine the two images together using bitwise or and from there obtain our final image which looks the same as 8. We thus have multiple instructions operating on a single image, which we then combine to obtain our final output.

4.4 MIMD Implementation

The MIMD approach involves combining the previous two approaches to parallelizing the filter. We first split up the image into multiple chunks once more. However, this time around, we assign half the processors to carry out Gx convolution on the image chunks, while the other half carries out the Gy convolution on the image chunks. Fro m here, we run the image combination algorithms in parallel as well. After completion of parallel work, we serially combine the two images with a bitwise xor like before and obtain the final image edges.

4.5 Timing

Using a timer function in the program, we timed the time it took for image break down, convolution, and image reassembling if need be. All algorithm parts were taken into consideration, except the Gaussian filter application. To obtain an accurate indication of parallel efficiency we carried out 250 tests for each program in order to obtain an accurate reading of the timing while accounting for standard deviation and parallel randomness. We then carry out statistical analysis to identify if a difference exists.

5 Results

5.1 Timing

We computed an unpaired t test to determine what the fastest approach is. We found that the SIMD had a statistically significant faster runtime. $p(a) = 0.0001$ and was roughly a factor of 15 faster than the other approaches. When comparing the MIMD and MISD approaches we found that the MIMD approach was faster than MISD with a $p(a)=0.0001$ once more showing a significant difference. Thus, the order of program execution from fastest to slowest is as follows: SIMD, MIMD, MISD.

5.2 Ease of implementation

Ease of implementation is an important factor for a multitude of reasons. The main reason we chose this as a variable is due to the importance of applying academic research in practical work environments. When we look at our implementation approach, the MISD approach was the simplest to implement. It required a simple splitting of the Gx Gy filters, which could then be easily recombined later with a bitwise xor. Following that, the implementation of SIMD was the second most simple approach. The benefit of applying SIMD to edge detection is that it naturally lends itself to the architecture. We have a couple simple for loops that operate with the same instructions on some given pixels - so really SIMD is just a matter of splitting up the pixels in the image and assigning them to processors. The difficulty arises in adding modular splitting as one may not have enough processors to cover the image totally so one would need to approach this with an update function of sorts. Lastly, the implementation of MIMD would be the most difficult. While it involves the combination of the two previous architectural approaches, it still requires extra work and extra math in order to properly allocate image chunks to each filter, Gx and Gy. From there, we need to carry out separate combinations of the two filters in order for final image chunks to have been properly convolved before the Gx Gy combination itself.

5.3 Quality of result

Overall, quality of result was similar across all images. This makes sense as ultimately the convolution that is carried out will be the same. One issue that we did run into was the bitwise XOR combination of Gx and Gy images led to a slightly more crisp looking output than the other outputs, but this can be attributed to the fact that bitwise XOR tends to have a slightly different, but higher quality output when compared to the classic formula of squaring both magnitudes and obtaining the summed values square root [8].

6 Discussion

6.1 On the Difference between Implementations

We saw that the SIMD implementation was the fastest by a factor of 15, with the MIMD being the second fast, and the MISD being third fastest. If we look at how the processors split up the work, this logically makes sense. The MISD approach is only able to utilise two processors as we assign the Gx and Gy filters to different processors, however, the rest of our cores are left utilised. When looking at previous research that explored implementation, we see that almost no papers take the MISD architectural approach for edge detection algorithm speed ups. When looking at the SIMD approach we see that it is the fastest algorithm we have designed. This makes sense because when we look at what GPU's are meant to be, they are really good at carrying out simple operations. Furthermore, as the GPU's must carry out only one type of instruction, this type of approach is perfect for GPU based execution. The strength here lies in that we can split the image apart into multiple chunks from where we can assign GPU cores to each chunk. We also would like to note that the serial bottleneck here is unavoidable. Due to the need for serial splitting of the image and recombination later down the line after convolution, that bottleneck cannot be parallelized and optimized away. Lastly, the MIMD approach is a combination of previous approaches, but it is slightly slower than SIMD. This can be attributed to the fact that we require more reassembling steps. When working with the chunk breakdown of the image, we obtain the same array of image chunks. When looking at the parallel part, MIMD and SIMD are similar in that we are utilising the same amount of cores as well - thus parallelization is similar in terms of time. The time difference though is attributed to the end step of reassmebling. Normally, in a SIMD approach you only need to reassemble the chunks and output the result. However, with our MIMD approach - we also need to bitwise xor the image, since we are using the Gx and Gy kernel seperately. Thus, MIMD contains an extra reassembling step, which needs to iterate through the whole image, subsequently causing the notable timing slow down.

6.2 Limitations

It is important to note a few limitations of this study. While the authors of this paper attempted their best at the implementation of the algorithms in different architectures, it is very plausible that we have not hit the most optimal algorithms for those architectures. Thus, while our results have demonstrated a significant difference between the efficiencies of the architectural approaches, with our logical backing making sense as well, it is important for future work to attempt different algorithms or provide proofs of optimalness to ensure that we have obtained the quickest algorithm.

6.3 Relation to Previous Work

Comparing our results to previous work done we note that most of the applications of GPU parallelization are SIMD approaches which demonstrate a considerable speedup improvement [27, 22, 26, 37]. Few papers approach this from a MIMD perpsective, with one paper [5] finding a slower speedup that the SIMD approach. The MISD approach is not found in literature for parallel speedup because of the limitations of GPU programming on only one piece of data. Our results support previous research and expand on it by clarifying the relation between taxonomical approaches in terms of speedup.

6.4 Future Work

Future work should focus on attempting to explore the MIMD taxonomy approach to edge detection. It seems that there is potential there, and the current research surrounding it does not obtain the highest efficiency algorithm. Obtaining such an algorithm could potentially lead to an improvement in the various algorithms utilising edge detection such as self driving cars or computer vision.

7 Conclusion

We carried out three different parallel taxonomical implementations to the same edge detection problem, using the Sobel edge detection algorithm. We compared the approaches on their timing, their ease of implementation and their effectiveness at generating an image. Our results showed that the SIMD approach was the most time effective to implement, with the MIMD approach being the second most quick, and with the MISD approach coming in last for this particular problem. Our results verify previous academic work in this area and demonstrate the need for further active research into applying MIMD solutions to parallel problems.

References

[1] Andrea Aprovitola and Luigi Gallo. Edge and junction detection improvement using the canny algorithm with a fourth order accurate derivative filter. In *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems*, pages 104–111. IEEE, 2014.

[2] Simon Baker, Shree K Nayar, and Hiroshi Murase. Parametric feature detection. *International Journal of Computer Vision*, 27(1):27–50, 1998.

[3] RJ BEATTLE. Edge detection for semantically based early visual processing. 1982.

[4] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[5] Taieb Lamine Ben Cheikh, Gabriela Nicolescu, Jelena Trajkovic, Youcef Bouchebaba, and Pierre Paulin. Fast and accurate implementation of canny edge detector on embedded many-core platform. In *2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS)*, pages 401–404. IEEE, 2014.

[6] Brijmohan Daga, Avinash Bhute, and Ashok Ghatol. Implementation of parallel image processing using nvidia gpu framework. In *International Conference on Advances in Computing, Communication and Control*, pages 457–464. Springer, 2011.

[7] Rachid Deriche and Thierry Blaszka. Recovering and characterizing image features using an efficient model based approach. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 530–535. IEEE, 1993.

[8] Adrian-Viorel Diaconu and Ion Sima. Simple, xor based, image edge detection. In *Proceedings of the International Conference on ELECTRONICS, COMPUTERS and ARTIFICIAL INTELLIGENCE - ECAI-2013*, pages 1–6, 2013.

[9] Piotr Dollár and C Lawrence Zitnick. Structured forests for fast edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1841–1848, 2013.

[10] A Downton and D Crookes. Parallel architectures for image processing. *Electronics & Communication Engineering Journal*, 10(3):139–151, 1998.

[11] Michael Flynn. *Flynn's Taxonomy*, pages 689–697. Springer US, Boston, MA, 2011.

[12] Estevão S. Gedraite and Murielle Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. In *Proceedings ELMAR-2011*, pages 393–396, 2011.

[13] Xiaochen He and Nelson Hon Ching Yung. Performance improvement of edge detection based on edge likelihood index. In *Visual Communications and Image Processing 2005*, volume 5960, pages 1664–1673. SPIE, 2005.

[14] Berthold KP Horn. The binford-horn line-finder. 1973.

[15] Zujun J Hou and Guo-Wei Wei. A new approach to edge detection. *Pattern Recognition*, 35(7):1559–1570, 2002.

[16] Manfred H Hueckel. An operator which locates edges in digitized pictures. *Journal of the ACM (JACM)*, 18(1):113–125, 1971.

[17] Manfred H Hueckel. A local visual operator which recognizes edges and lines. *Journal of the ACM (JACM)*, 20(4):634–647, 1973.

[18] Ramesh Jain, Rangachar Kasturi, Brian G Schunck, et al. *Machine vision*, volume 5. McGraw-hill New York, 1995.

[19] James Lee, R Haralick, and Linda Shapiro. Morphologic edge detection. *IEEE Journal on Robotics and Automation*, 3(2):142–156, 1987.

[20] De-Sian Lu and Chien-Chang Chen. Edge detection improvement by ant colony optimization. *Pattern Recognition Letters*, 29(4):416–425, 2008.

[21] David Luebke. Cuda: Scalable parallel programming for high-performance scientific computing. In *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 836–838, 2008.

[22] Yuancheng Luo and Ramani Duraiswami. Canny edge detection on nvidia cuda. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[23] Xiaoju Ma, Bo Li, Ying Zhang, and Ming Yan. The canny edge detection and its improvement. In *International Conference on Artificial Intelligence and Computational Intelligence*, pages 50–58. Springer, 2012.

[24] Vishvjit S Nalwa. Edge-detector resolution improvement by image interpolation. *IEEE transactions on pattern analysis and machine intelligence*, (3):446–451, 1987.

[25] Vishvjit S Nalwa and Thomas O Binford. On detecting edges. *IEEE transactions on pattern analysis and machine intelligence*, (6):699–714, 1986.

[26] Shengxiao Niu, Jingjing Yang, Sheng Wang, and Gengsheng Chen. Improvement and parallel implementation of canny edge detection algorithm based on gpu. In *2011 9th IEEE International Conference on ASIC*, pages 641–644. IEEE, 2011.

[27] Kohei Ogawa, Yasuaki Ito, and Koji Nakano. Efficient canny edge detection using a gpu. In *2010 First International Conference on Networking and Computing*, pages 279–280. IEEE, 2010.

[28] Frank O'gorman and MB Clowes. Finding picture edges through collinearity of feature points. *IEEE Transactions on computers*, 25(04):449–456, 1976.

[29] Victor Podlozhnyuk. Image convolution with cuda. *NVIDIA Corporation white paper, June*, 2097(3), 2007.

[30] Judith MS Prewitt et al. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970.

[31] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids*. 01 1963.

[32] Azriel Rosenfeld and Mark Thurston. Edge and curve detection for visual scene analysis. *IEEE Transactions on computers*, 100(5):562–569, 1971.

[33] Jun Shen and Wei Shen. Image smoothing and edge detection by hermite integration. *Pattern Recognition*, 28(8):1159–1166, 1995.

[34] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[35] George E Sotak Jr and Kim L Boyer. The laplacian-of-gaussian kernel: a formal analysis and design procedure for fast, accurate convolution and full-frame output. *Computer Vision, Graphics, and Image Processing*, 48(2):147–189, 1989.

[36] Tao Yang and Yue-hong Qiu. Improvement and implementation for canny edge detection algorithm. In *Seventh International Conference on Digital Image Processing (ICDIP 2015)*, volume 9631, pages 99–108. SPIE, 2015.

[37] Nan Zhang, Yun-shan Chen, and Jian-li Wang. Image parallel processing based on gpu. In *2010 2nd international conference on advanced computer control*, volume 3, pages 367–370. IEEE, 2010.

[38] Huili Zhao, Guofeng Qin, and Xingjian Wang. Improvement of canny algorithm based on pavement edge detection. In *2010 3rd International Congress on Image and Signal Processing*, volume 2, pages 964–967. IEEE, 2010.

[39] Djemel Ziou, Salvatore Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998.