Exploring Parallel and Anytime SIFT

By Kevin Tang
Supervised By: Professor John Korah
California Polytechnic State University Pomona, College of Science

Abstract:

Scale Invariant Feature Transform (SIFT) is a computer vision algorithm with widespread applications from geospatial sciences to 3D modeling to robotics. It is used to extract independent features of an image that are identifiable in other images despite changes in scale, noise, illumination, and to some degree rotation. We researched and explored ways of improving the SIFT algorithm by attempting a design that is both Anytime and makes use of parallel processing. As undergraduate students much of our time was spent seeking to understand and implement the SIFT algorithm, some analysis of the parallel aspect was completed, and some theories for an Anytime implementation proposed.

Introduction:

SIFT is a robust feature matching algorithm that eliminates the need for manual work in tasks such as stitching, mosaics, alignment, and matching of images among others. [11] SIFT is also well known within the scientific community and produces the best results for the amount of computational effort in comparison to other feature detectors. In the context of image retrieval, there is demand for an efficient and flexible method of processing and indexing new entries into a database of images for matching and retrieval based on its characteristics. Image indexing for search applications is costly and machine learning implementations are sometimes outperformed by procedural feature matching algorithms such as SIFT. Additionally, image databases do not always have a steady influx of new images to be indexed and archived. Oftentimes many images are dumped at once, or there are long periods with few new entries. The objective of this research is to explore implementations of SIFT that can confront these challenges.

In order to extract features from an image that are invariant to scale, translation, rotation, and lighting SIFT generates a scale space and computes a difference of Gaussians. This is the first step in the SIFT algorithm, and the one which is most explored in our research.

- The original image is scaled down to half its rows and half its columns. The resulting image is scaled down again repeatedly to generate a specified number of scales (octaves). This step has the effect of creating invariance with regards to the scale of the image when identifying its features.
- Each scale of the image is blurred several times using a Gaussian Box Filter, creating several duplicates of each at different blur levels. This step serves to reduce noise in the image, and focus on the outlines of objects or shapes that we wish to identify as features.
- At each scale, the pixel values of each image are subtracted from their neighbors in terms of blur level, producing a difference of Gaussians. The outputs of this step often

appear to be dotted outlines of key features of the original image, preserving only what's most prominent about them.

We focus on the optimization of these processes as the foundation for an efficient and flexible SIFT implementation.

Background:

Several speedup factors have been calculated for SIFT on different technologies. [3]There have even been specific hardware architectures designed specifically for SIFT computation. Such architectures are of course more efficient and powerful for feature extraction. [8] Our implementation of SIFT, however, is geared toward the Cal Poly Pomona High Powered Computing environment, a compute node cluster of twenty with no special or dedicated hardware. This research assumes execution on consumer-available hardware which is more accessible.

Four main components of the SIFT algorithm have been identified as the most computationally expensive: Building the Gaussian Scale Space, Keypoint Detection and Localization, Orientation assignment and Keypoint Description, Matrix Operations [6]. As such our research prioritizes the acceleration of these components first. Several possible configurations of Building the Gaussian Scale Space in parallel are discussed later.

Not only can the process of SIFT be parallelized, but the input data as well. In cases of large images, it becomes difficult to parallelize SIFT as whole images no longer fit into individual compute node caches. [2] Although this paper focuses on fine grained parallelism of the SIFT process, in consideration of large images it is necessary to divide into subimages as a form of high level data parallelism. Additionally, less data is lost if images are divided two-dimensionally as opposed to one-dimensionally. It is simpler to divide an image between compute nodes by row or by column only, and extra calculations must be performed and stored in order to determine the coordinates of the subimage for each compute node. In geospatial and radar applications, images to be processed are too large by consequence of subject matter, [1][2] but the simple purpose of image retrieval will often not require us to divide our input data. Despite this, it may be more efficient to divide even images that do not reach a size threshold for the sake of speed. This may especially be useful in case of idle processors.

Methodology:

As undergraduate students, our research began with lectures on image retrieval, indexing, and the SIFT algorithm. Much time was dedicated to making sure there was a clear understanding of SIFT before beginning implementation. Groups were formed which were dedicated; one to understanding the blurring aspect of SIFT, and two to understanding the scaling aspect. Progress was made by both teams, and information exchanged via team presentations which deepened our knowledge of SIFT as a whole.

Once there was a clear understanding of the initial phases of SIFT, methods of parallelizing it were proposed. We hypothesized first to divide the work along the scaling axis. Separate scales were to be calculated on separate nodes, on which the blur levels would be calculated in series and then subtracted from one another. This process seemed the most intuitive for a few reasons:

1) Data from downsampling the images could be reused by the next scale. Each scale could build from the last, leading to less redundant processing.
2) Attempting to parallelize along the blurring axis leads to more scaling calculations and less blurring calculations, when scaling calculations are more expensive.
3) Blurring of each octave is local, and no data transfer is needed to calculate the difference of Gaussians.

Similar designs for concurrent SIFT implementations have been tested to significant speedups, however with several other optimizations upon the original SIFT algorithm. [7] However, upon further inspection this design suffers heavily from load imbalance, as the smaller scales will complete their blur and DoG calculations much earlier than the larger scales. Also, the dependency of each scale upon the last introduces data transfer time between the scales.

To solve these issues, we proposed to parallelize the algorithm along the both the blurring and scaling axes.

1) Each node is given a copy of the original image via tree broadcast.
2) Each node scales and blurs their image to the proper level independently.
3) Nodes identify their incrementally blurred neighbors and send their processed image.
4) All nodes except those without an incrementally blurred neighbor calculate a DoG

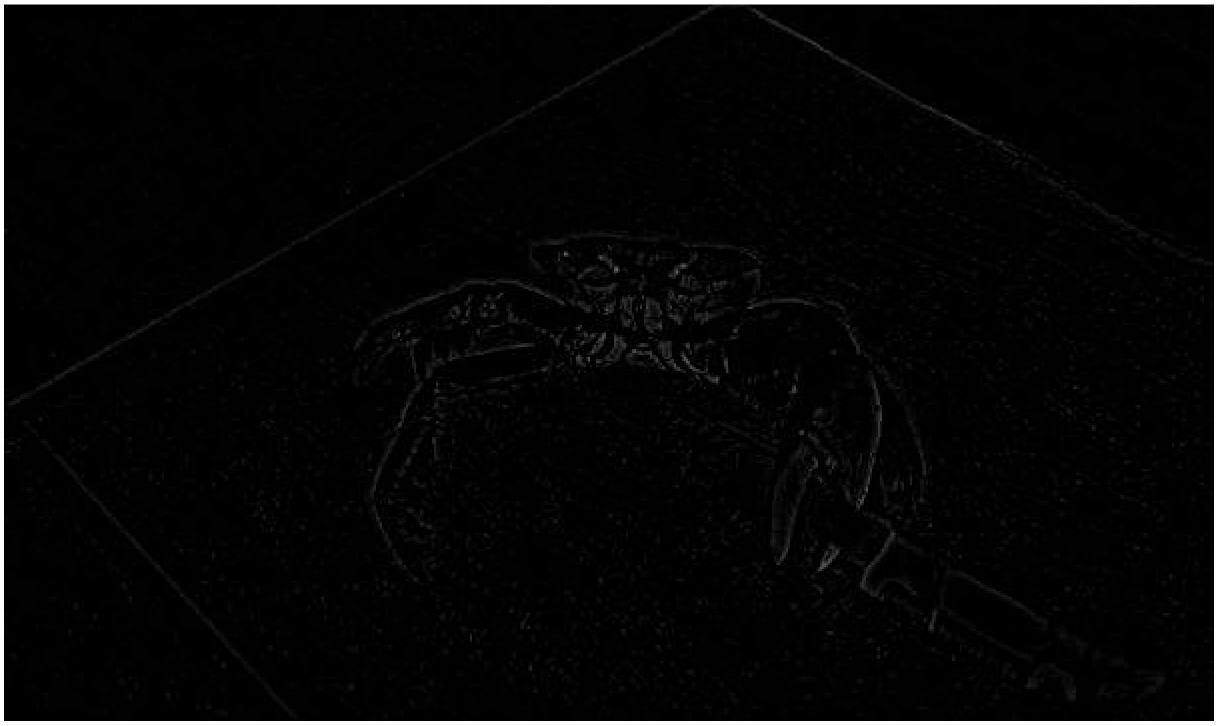
Although this method leads to many redundant scale calculations, no node is dependent on another node to perform them. In the previous design a node might have to wait for its predecessor to scale and then transfer the image, but in this design it calculates the initial scaling itself, removing the communication time. While we still suffer from load imbalance it’s significantly less than when all the blur levels were calculated in series, as speed differences were multiplied by the number of blurred images at each scale. This design emphasizes minimal communication time and independence between nodes which causes the same scale calculations to be performed on each of the nodes instead of the results being propagated. Although we will get DoGs sooner than if we were to propagate the scaled images, several computation cycles will be dedicated to redundant calculations.

For this a third algorithm is theorized in which we seek to minimize redundancies. In such a design, idle time would be introduced while waiting for data to propagate. It may be possible to repurpose these idle clock cycles for other calculations, potentially pipelining the next image to be processed.

First Parallel Implementation:

A demonstration of Gaussian Scale Space Generation was designed by me for the purpose of understanding the initial stages of SIFT and how to make them execute concurrently. It makes use of Message Passing Interface, a popular Application Programming Interface which allows intercommunication between processes or nodes of the same cluster. It also employs OpenCV computer vision libraries which provide image data structures and Gaussian Scaling and Blurring functions. The demonstration is a C++ program designed for use on the Cal Poly Pomona HPC cluster. It first loads the image for which it will generate the scale space and difference of Gaussians on three compute nodes. Then, using MPI functionality, each node identifies its rank or ID. The image is scaled based on this ranking and a new array of four images is created. Each image is blurred three times, each time incrementally more than the last and stored in the array along with the original. Finally, the images in the array are subtracted from their neighbor and sent back to the root node to be compiled in an array of nine DoGs.

The original image, from a 2016 viral video.



A sample DoG result from the demo.

Runtime Analysis:

For time analysis for this problem, we define the following terms:

n: the number of rows of pixels in the image
m: the number of columns
k: the length and width of the kernel for Gaussian Blurring, usually 5
b: the number of blur levels
s: the number of scale levels
G: the bandwidth
P: the number of processors

In these terms, the cost of a Gaussian Blur for an image would be:

$$(nm) * 2k^2$$

n × m is the pixel count, on which must be performed a Gaussian Blur, which is a form of weighted averaging over the kernel 5 × 5, or k^2. To weight and average the pixel values, it costs one addition and one multiplication for each member of the kernel, so $k^2 \times 2$.
Scaling the image is done using a Gaussian Blur as well, so it has the same initial cost as blurring:

$$(nm) * 2k^2 + \tfrac{3}{4}(nm)$$

What's added to the end is the cost of removing three quarters of the pixels of the original image, since we are scaling the image down by half its rows and half its columns every time. For all blur and scale levels, this becomes:

$$\sum_{i=1}^{s} b * (\tfrac{1}{4})^{i-1} * (2nmk^2) + (\tfrac{3}{4})^{i} nm$$

We find the closed form of this by the following formula:

$$\sum_{k=x}^{y} a^k = \frac{a^x - a^y}{1-a}$$

Which gives us:

$$\frac{4 - 4(\frac{1}{4})^s}{3}\ (b)(nm2k^2)\ + (\tfrac{9}{4} - 4(\tfrac{3}{4})^{s+1})nm$$

If all steps were to be performed in series, this would be the amount of time it would take. This also serves as the control for how fast of a speedup we get from parallelizing. Let's compare it to parallelizing along the scaling axis as in our demo, k=5, b=4, s=3, P=3:

$$\sum_{i=1}^{b} 2nmk^2 = 2bnmk^2$$

This is the performance of the node that will take the longest to complete, which is the one that handles the blurring of the original scale image. Divided by the runtime of the serial algorithm:

$$\frac{8*25nm}{\frac{5}{16}(8nm*25)+\frac{9}{4}-(\frac{81}{256})nm} = 0.756333554915...$$

We shave off a modest quarter of the runtime in the demo implementation. If we were to parallelize this by dividing up the image, we would see drastic improvements. In the case we have double the processors, 6, and each new node handles one half of the work of the previous nodes, we would be twice as fast at around 0.37 times the runtime of the serial algorithm.

Proposed Anytime SIFT:

During the creation of the scale space, a minimal number of blur and scale levels must be created in order to receive sensible results. There must be at least four levels of blur in order to produce three DoG at each scale, and there must be at least three levels of scale. This is to produce a 3x3 matrix of DoG which is the minimal matrix required for calculating keypoints. However, it is likely that matrices larger than 3x3 produce both a greater quantity and quality of keypoints. For the sake of a more robust feature extraction, it may be preferable to generate more blur and scale levels despite increasing the workload.

To create an Anytime SIFT algorithm, we propose that when there is little that needs to be done, more scale and blur levels should be generated up to some point of diminishing returns. When there is a large influx of new images, the algorithm generates minimal amounts of blur and scale levels. These “good enough” blur and scale levels can be stored alongside the image that is being indexed, and added to later when the workload is less. In this way the algorithm could be flexible to demands in image retrieval applications.

References:

[1] Quan Sun, Jianxun Zhang. Parallel Research and Implementation of SAR Image Registration Based on Optimized SIFT. TELKOMNIKA Indonesian Journal of Electrical Engineering. 2014.

[2] Stanislav Bobovych. Amy Apon, Parallelizing Scale Invariant Feature Transform On A Distributed Memory Cluster. INQUIRY. 2011.

[3] Miaoqing Huang, Chenggang Lai. PARALLELIZING COMPUTER VISION ALGORITHMS ON ACCELERATION TECHNOLOGIES: A SIFT CASE STUDY. Department of Computer Science and Computer Engineering University of Arkansas.

[4] Junchul Kim, Younghan Jung, Xuenan Cui, Hakil Kim. A Fast Feature Extraction in Object Recognition Using Parallel processing on CPU and GPU. SMC. 2009.

[5] Dongdong Yu, Feng Yang, Caiyun Yang, Chengcai Leng, Jian Cao, Yining Wang, Jie Tian. Fast Rotation-Free Feature Based Image Registration Using Improved N-SIFT and GMM Based Parallel Optimization. IEEE. 2015.

[6] Qi Zhang , Yurong Chen, Yimin Zhang, Yinlong Xu. SIFT Implementation and Optimization for Multi-Core Systems. Dept. of Computer Science, Univ. of Science and Technology of China Intel China Research Center, Intel Corporation. 2008.

[7] Liang-Chi Chiu, Tian-Sheuan Chang, Jiun-Yen Chen, Nelson Yen-Chung Chang. Fast SIFT Design for Real-Time Visual Feature Extraction. IEEE TRANSACTIONS ON IMAGE PROCESSING. 2013.

[8] Murad Qasaimeh, Assim Sagahyroon, Tamer Shanableh. A Parallel Hardware Architecture for Scale Invariant Feature Transform (SIFT). Department of Computer Science and Engineering American University of Sharjah.

[9] Anton I. Vasilyev, Andrey A. Boguslavskiy, Sergey M. Sokolov. Parallel SIFT-detector implementation for images matching. Keldysh Institute of Applied Mathematics of Russian Academy of Sciences.

[10] Donglei Yang, Lili Liu, Feiwen Zhu, Weihua Zhang. A Parallel Analysis on Scale Invariant Feature Transform (SIFT) Algorithm.

[11] Tinne Tuytelaars, Krystian Mikolajczyk. Local Invariant Feature Detectors: A Survey. Foundations and Trends(R) in Computer Graphics and Vision. 2008.