Hyper Parallel Image Feature Extraction Applications on Modern Multicore Processors

T. Mogale
Department Of Computer Science
North West University
Mafikeng Campus
Private Bag X2046, Mmabatho 2735, South Africa
Hope.Mogale@ieee.org

M.B Esiefarienrhe
Department Of Computer Science
North West University
Mafikeng Campus
Private Bag X2046, Mmabatho 2735, South Africa
eseifabukohwo@gmail.com

***Abstract*—Image Feature Extraction (IFE) Algorithms are the most commonly algorithmd utilized in many modern Image Processing Applications. Most modern image feature extraction methods involve the use of a multi-stage algorithm to detect edges in a wide range of images. Edge detection is at the forefront of image processing and hence, it is crucial to have at an up to scale level. Multicore Processors have emerged as the next solution for tackling compute intensive tasks that have a high demand for computational power on personal computers. However, most IFE algorithms of today under utilize these processors and majority of their cores always remain unused and this results in a phenomenon called dark silicon. With significant changes that restructured the microprocessor industry, it is evident that the best way to promote efficiency and improve performance on modern microprocessors is no longer by increasing the clock speeds on traditional monolithic processors but by adopting and utilizing Processors with parallel multicore architectures. In this paper we provide a hyper parallel implementation of Image Feature Extraction applications on Multicore Processors. We use a case of Canny Edge Detector as an example to demonstrate that with the right approach IFE applications can scale well on modern parallel Architectures. We show that by injecting hyper parallelism such applications reduce under utility of modern parallel architectures such as multicore processors. Our results show significant improvements in Speedup rates, CPU Usage Over Wall Clock Time, CPU Usage Per Core, and CPU Context Switch Per Core. Overall this proves that injecting Hyper Parallelism on compute intensive applications such as ones involving IFE can save time, help improve productivity, and diminish problems that result from under utility of parallel computing resources such as dark silicon.**

Keywords - ***Multicore Processors, Parallel Patterns, Efficiency, IFE, Algorithms and CED***

I. Introduction

Following the dominant change of technology in the microprocessor industry, processors have evolved eminently from the traditional Monolithic single-core processors to Multicore Architectures which feature Superscalar capabilities, Multithreading, and Advanced Vector Extentions (AVX) enabling Single Instruction Multiple Data (SIMD) technology. The unbated Moore's Law continues to bestow performance in the multicore era [1]. The evolution of Multicore processors was sparked by the impacts of limits encountered by microprocessor developers over the years. Several of these limits could be ignored however the most notable ones that architects could not ignore are the ones discussed by authors in [2] known as the three walls. The first of these walls to be encountered or realized was the power wall as a result of unacceptable growth in power usage with clock rate and also the realization was that above around 130W air cooling is not sufficient[3], [2]. Second was the Instruction-level Parallelism (ILP) wall due to the limits at which one can achieve low-level parallelism. The last wall is the memory wall which resulted because processor speeds were highly discrepant to memory speeds. The most significant of all of the three walls was the power wall. This had so much impact in the microprocessor industry and perhaps the most compelling factor for architects to shift processor designs to multicore architectures and this successfully led to the establishment of the Multicore processor era.

A. Hyper Parallelism

Hyper Parallelism is a form of parallel computation involving extensive use of Algorithmic Skeleton Frameworks(ASkF). Since the shift from traditional monolithic microprocessors to modern multicore microprocessors became predominant in most computer systems of today. The prevalent change in chip architecture simply meant that for application developers to scale applications improve performance they would have to adopt the legacy style of development that was popular among parallel computer developers known as parallel programming. Parallel programming although effective, appears not to be mundane to many developers hence, why the application development industry saw much of resistance towards adopting it over the years. Further there is a lack of a defined structured methodology for application developers to follow or port previously developed code to utilize Multicore platforms and parallel architectures. Many of the existing developed applications still continue to underutilize parallel architectures. To aid this and realize hyper parallelism we adopt a universal parallel computational model we call the Golden Circle of Parallelism (GCP) that we will use to define our structured approach and to realize Hyper Parallelism. The GCP Model is composed of three layers and these layers are structured hierarchically as Shell, Kernel and the Core. All the three

layers of the GCP Model help realize hyper parallelism. We shall utilize the GCP as a tool to realize hyper parallelism on IFE Applications such as the Canny Edge Detector.

B. Canny Edge Detector an Application of IFE

Canny Edge Detector (CED) is a operator used commonly for image feature extraction and also adopted by many image processing algorithms. This operator involves the use of a multi-stage algorithm to detect a wide range of edges in images. CED named after its author [4] nominates a computational approach to edge detection.

Canny Edge Detector operator mainly aims at achieving:

1) **Low error rate** - Reliable for accurate detection of only existent edges. For low error rates which yields good detection canny edge detector uses Signal-To-Noise (SNR) ratio and its criterion for low error rates [5] on detection is:

$$SNR = A\left|\int_{-T}^{0} f(x)dx\right| \Big/ \left(\sigma\sqrt{\int_{-T}^{T} f^2(x)dx}\right)$$

2) **Good localization** - The distance between edge pixels detected and real edge pixels have to be minimized. The criterion for good localization [5] is defined as:

$$L = 1 \Big/ \sqrt{\int_{-T}^{T} y^2 \Pr(y)dy}.$$

3) **Minimal response** - Restrict only one detector response per edge. In other terms the detector should produce multiple maxima. According to Canny [4] the minimal response criterion is defined by:

$$x_{\max}(f) = 2\pi\sqrt{\int_{-T}^{T} f'^2(x)dx} \Big/ \int_{-T}^{T} f''2(x)dx$$

Because of the aforementioned attributes CED has been widely adopted for image processing applications that involved edge detection. Edges of an image are important in determining features of digital images has why CED has been applied in many areas of image feature extraction. For enhanced edge detection, CED allows additional algorithms such as Sobel algorithm to be employed on the multi-stages of CED.

In this paper we aim to provide a hyper parallel implementation of CED that can effectively scale on multicore processors. The rest of the paper is organized as follows. Section II provides a survey of related work. In section III the synopsis and methodology is described in detail. Section IV gives a detailed discussion of the obtained results. Finally sectionV briefly outlines the conclusion.

II. Related Work

There has been substantial amount of research that accounts for CED and its application to image processing operations published by scholars in the past and recently.

Ali et al [6] implemented CED for feature extraction on remote sensing images and recommended CED as an enhancement tool that can be used for troublesome remote sensing images that can be corrupted by point noise. As illustrated by the authors in [7] IFE is a time consuming process and this is even worse when the images to be processed are in large quantities of if the image has high quality. Consequently this is often encountered by image processing applications on the internet because of the high frequency of image data on the internet. Researchers in [8] have identified that the calculation of IFE algorithms constantly increases, and this contributes the most time consuming step in image steganography detection. Researchers in [7] discovered that most IFE methods do not care much about performance and do not take note on the utilization of the highly developed microprocessor architectures. Almost all image processing applications are implemented serially and this leads to poor results in terms of performance even on highly developed microprocessor architectures of the Modern day computer systems.

Researchers in [9] surveyed existing shape-based feature extraction. Yang and team recommended that efficient shape features must present essential properties such as identifiability, translation and noise resistance among others. They further outlined that in a simple form a shape descriptor is simply a set of numbers that describe a given shape feature, and in one of the requirements of a shape descriptor they state that the computation of distance between descriptors should be simple; otherwise execution time will present overhead. Since the descriptor operates in serial and is not optimized for multicore architectures this may still be prevalent on application making use of the descriptor.

Kornaros et. al [10] explored several microarchitectural alternatives to improve performance for edge detecting algorithms and they proposed reconfigurable multicore prototype which was able to achieve 5x speed up rates. Hao and team in [11] successfully parallelized a Scale Invariant Feature Transform (SIFT). In their work they state that in order to meet computation demands they optimized and parallelized SIFT to accelerate its performance on Multicore Architecture systems. Furthermore they indicate that SIMD integrated with Multicore Architectures bring an extra 85% performance increase. Luo et. al [12] successfully implemented CED on NVIDIA Compute Unified Device Architecture (CUDA) and this shows that CED can also be implemented on GPU platforms which is relevant because multicore processors have GPU unit. Zhang et. al [13] presented an improved parallel SIFT implementation which is able to process video images in real-time utilizing multicore processors and the results showed great improved in terms of speedup in comparison to GPU implementation. Cho and team in [14] spearheaded a study that construed the key factors used in the design and evaluation of image

processing algorithms on massive parallel platforms. Clemons et. al [15] presented an embedded multicore design named EFFEX with novel functional units and memory architecture support capable of increasing performance on mobile vision applications while lowering power consumption rates.

III. Synopsis and Method

A. Synopsis

We have chosen a set of images ranging from different fields of research. Our hardware configuration consists of two multicore processors listed on table 3.1. We have fully parallelized the CED algorithm to take full advantage of the available resources on any given processor. Parallel patterns included with Cilk Plus have been applied on the gaussian filter and on Sobel's algorithm which finds the intensity gradient of the image.

Table 3.1 Hardware Configurations

Processor	Vendor	Core Count	Clock Speed
Core i3	Intel	2cores, 4 CPUs	3.4 GHz
Core i7	Intel	4cores, 8 CPUs	3.4 GHz

Abiding with Amdahl's law the hysteresis part of the CED algorithm has been noted to contribute unparalleled work because of the serial elision it creates.Our developed CED algorithm was implemented on Microsoft Visual Studio extended with OpenCV and Intel Cilk Plus and was applied on different applications of different disciplines to see how it performs. For all these research areas both the optimized algorithm and the non optimized of CED has been carried out on the images and the results were recorded for each.

B. Parallel Canny Edge Detector

The canny edge detector algorithm we describe here is well known and has been widely adopted and used in many fields of image processing. This edge detector was first described by its author in [4] illustrated in figure 3.1 and has been improved and modified by researchers over time most notably authors in [5].

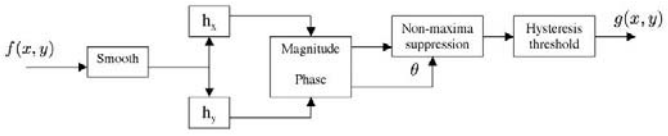

Figure 3.1 Illustration of Canny Edge Detector[5]

Pragmatically we describe the parallel implementation of the Canny Edge Detector below. The formal description is defined in algorithm 1 listing.

1) **Filter out any noise**
   Apply parallel patterns to the Gaussian noise filter.
2) **Find the intensity gradient of the image**
   Employ Sobel Algorithm with parallel modifications:
   a) *First apply parallel patterns to a nominal pair of convolution masks (Gx, Gy)*

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

   b) *then compute the gradient strength and its direction with parallel computation*

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

3) **Now begin to apply Non-Maximum Suppression. This is to remove pixel that are not part of the edge**
4) **Finally perform Hysteresis:**
   a) *If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge*
   b) *If a pixel gradient value is below the lower threshold, then it is rejected.*
   c) *If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.*

C. Parallel Patterns and Programming Model

For the parallel implementation of CED we have chosen Intel Cilk Plus which provides composable parallel patterns [16] that guarantee determinism. Cilk Plus is a multithreaded language that uses a work stealing scheduler that has the ability to distribute work loads evenly on the cores of the multicore processor. Cilk is highly algorithmic [2] [17] hence, it provides algorithmic skeletons that renders the Cilk runtime to take care of details such as load balancing, resource communication, and etc.

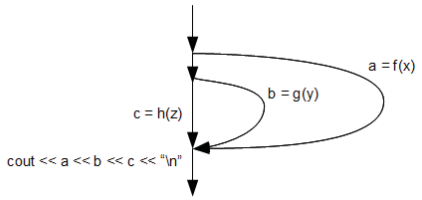

Figure 3.2 Direct Acyclic Graph (DAG)

Cilk plus features a set of parallel patterns highly suitable for numerical computation. The parallel patterns available in Cilk are the Map pattern, Fork Join, Reduction, and Reducers seen in figure 3.3. In addition the patterns are implemented explicitly in Cilk Plus using three keywords that form an extension of C/C++ and these are instrumental for structured parallelism. The keywords are `cilk_spawn` a fork-join pattern, `cilk_for` a thread-parallel map pattern, and

`cilk_sync` issues signals for map pattern. For the execution depicted on the DAG in figure 3.2 if the function *g(y)* was spawned it should not be assumed that both *g(y)* and *f(x)* will run in parallel because `cilk_spawn` only recommends what can run in parallel. Thread-parallelism in Cilk Plus is expressed using the notion of a strand.

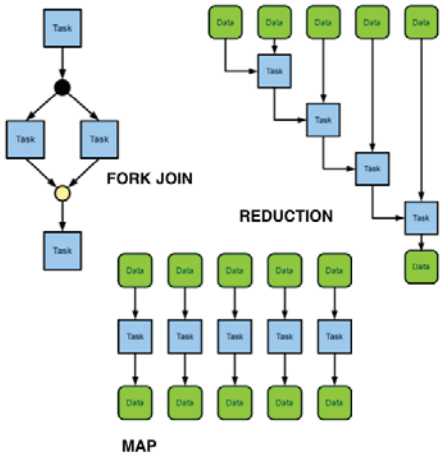

Figure 3.3 Cilk Parallel Patterns [2]

Algorithm 1 Canny Edge Detector with Cilk Parallel Patterns

```
1:  procedure cilk_spawn gaussianfilter(noise x)
    ▷ detect the remaining edges after filtering x
2:      for i = 1 → k do
3:          if f_i(x)%N == 0 then
4:              return
5:          end if
6:      end for
        ▷ Employ Sobel algorithm
7:      for i = 1 → k do
        ▷ Parallel loop (cilk_for)
8:          if threshold p >0 then
9:              p ← G_(x,y)
10:             θ ← arctan (G_(x) ÷ G_(y))
11:         end if
12:     end for
        ▷ Perform hysterisis
13: end procedure
```

Listing 1. Parallel CED Algorithm

D. CED Applications

We now apply the parallel CED on the images from different fields of research. As can be seen in figure 3.2 the increase in the threshold the CED filters more noise from the image. The CED algorithm is highly suitable for medical research applications and it can be very instrumental on applications of virology which involve highly virulent viruses.

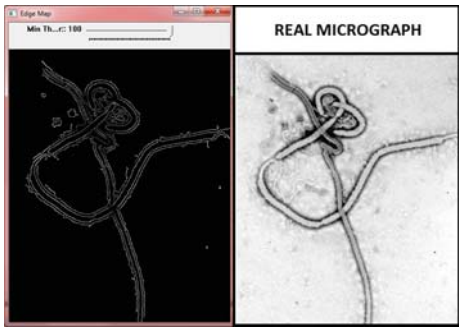

Figure 3.4 - CED Applied on Medical Research

We apply CED on computer vision applications and the operator yields the output result seen on figure 3.5 This would be very helpful in identifying different objects that the computer may encounter using computer vision.

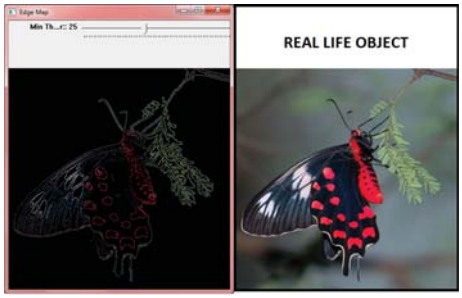

Figure 3.5 - CED Applied on Computer Vision

For example if the computer has a database containing characteristics, components and attributes of different objects then it can be able to scan and identify the object using the CED seen in figure 3.5. From the figure 3.6 it can be seen that the CED also performs well on Photography.

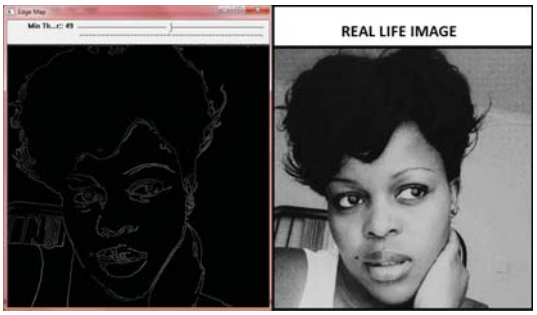

Figure 3.6 - CED Applied on Photography

IV. Results and Discussion

From experiments carried on the various images using both parallel(optimal) and non-parallel(suboptimal) Canny Edge Detector the following was measured and observed:

A. CPU Sampling And Total Usage

1) Suboptimal implementation: The sampling method used collects profiling data for every 10,000,000 processor cycles and this is very useful for detecting performance issues. For the suboptimal(non-parallel) or non-optimal implementation

of CED algorithm the profiler collected about 8,992 samples. A graphical of representation of CPU usage over wall clock time in seconds is shown in figure 4.1. From this graph a low CPU usage can be seen which is not ideal for application performance and user experience. The low CPU usage is in response to the total sample count.

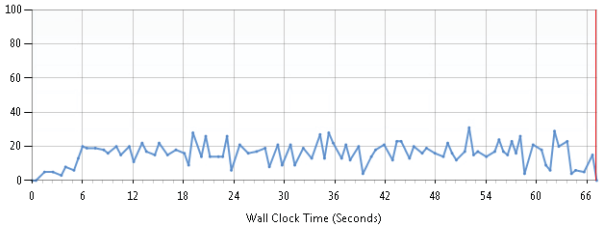

Figure 4.1 - Suboptimal CPU Usage Over Wall Clock Time

2) Optimal implementation: For the optimal implementation of the CED algorithm the profiler collected about 34,884 samples. In Figure 4.2 the graphical representation of CPU usage over wall clock time in seconds can be seen for the fully Optimized CED algorithm. This graph shows better CPU usage which is ideal for application performance and user experience. The efficient CPU usage is also in response to the total sample count.

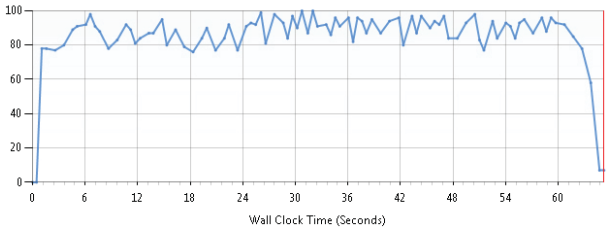

Figure 4.2 - Optimal CPU Usage Over Wall Clock Time

From the observed results it can be seen that the optimal CED out performs the suboptimal implementation of the CED algorithm. The above results provides only information for total CPU Usage in percentage. To be precise the total usage per core must be observed. Next we observe total usage per core for 4 core cpu and 8core cpu to test the developed CED for scalability.

B. *Total CPU Usage Per Core*

1) Suboptimal implementation (4 CPUs): To be precise if the CED algorithm is stable and durable in terms of performance total usage per core must be observed. In figure 4.3 this can be seen precisely, the total CPU usage per core for the suboptimal implementation. From the figure it can be seen that the utilization is uneven hence, declaring that some cores may be idle while others are working. This is not ideal for overall application performance on a Multicore Architecture system.

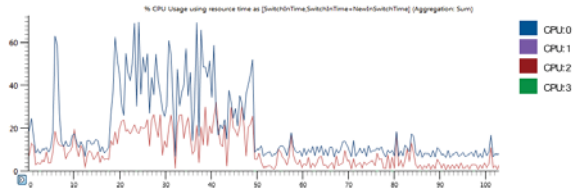

Figure 4.3 - Suboptimal CPU Usage Per Core(4 CPUs)

2) Suboptimal implementation (8 CPUs): In figure 4.4 we also show the suboptimal implementation of CED with no parallelism. From the figure it can be seen that this implementation is not ideal for a Multicore system with more CPUs.

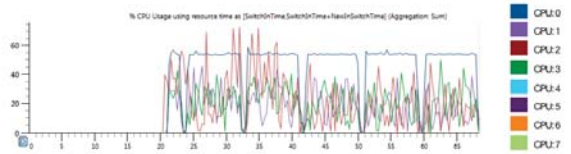

Figure 4.4 - Suboptimal CPU Usage Per Core(8 CPUs)

3) Optimal implementation (4 CPUs): The total usage per core for the Optimal implementation of CED on 4 cpu multicore processor is shown figure 4.5 and it can be see that there is an ideal usage per core. It can be seen that utilization per core is even and well distributed among the cores. This is mainly because of the work stealing scheduler provided by Cilk Plus runtime. This even distribution is ideal for Multicore Architecture based systems and will enhance overall application performance and user experience.

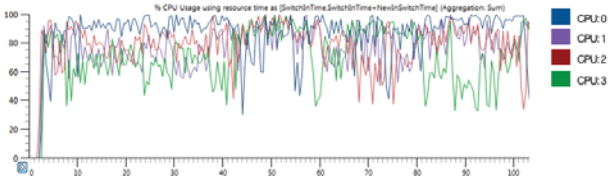

Figure 4.5 - Optimal CPU Usage Per Core(4 CPUs)

4) Optimal implementation (8 CPUs): To test if our CED implementation is scalable we have implemented it on the 8 CPU multicore processor and the results can be seen in figure 4.6. The results show ideal usage that fully maximizes and utilizes available CPU resources. The results seen in figure 4.6 serve as proof that our parallel implementation of CED is fully scalable for multicore processors.

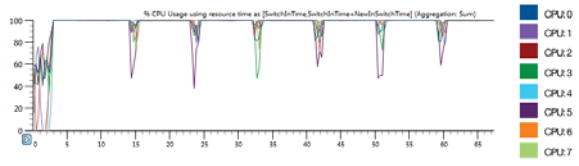

Figure 4.6 - Optimal CPU Usage Per Core(8 CPUs)

C. *CPU Context Switch Per Core*

1) Suboptimal implementation (4 CPUs): Context Switches are essential because they can affect the performance of Multicore processor system. Context switches occur when the operating system kernel switches the CPU between threads. Hence, in a Multicore Architecture system if the kernel only switches one core out of four between threads it can increase competition for CPU and raise tension which can degrade overall performance of the system.

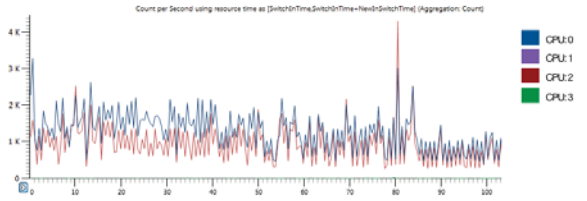

Figure 4.7 - Suboptimal Context Switch Per Core (4 CPUs)

2) Suboptimal implementation (8 CPUs): From figure 4.7 and figure 4.8 it can be seen that the kernel switches roughly two thirds of cores out of the remaining cores and hence, this explains why there is high context switch rates which may create tension and degrade overall system performance in the suboptimal implementation of CED.

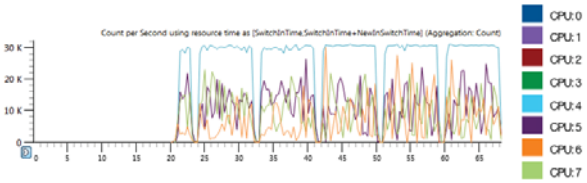

Figure 4.8 - Suboptimal Context Switch Per Core (8 CPUs)

3) Optimal implementation (4 CPUs): Context Switch Count for the optimal implementation of the CED can be seen in figure 4.9 for 4 CPUs and figure 4.10 for 8 CPUs. it is evident from the figures that the kernel switches the CPU at a full swing. In addition to this it is evident that the switch rate is relatively low and hence, it does not compromise the performance of Multicore system.

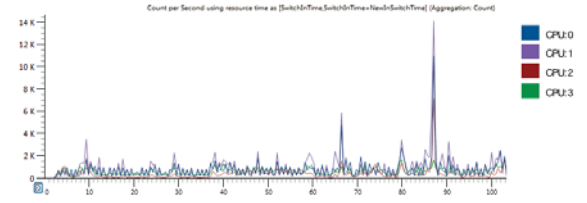

Figure 4.9 - Optimal Context Switch Per Core (4 CPUs)

4) Optimal implementation (8 CPUs): From figure 4.10 we can observe that the context switch rate is ideal. However, from figure 4.9 there are occasional high peaks displayed on the graph. These occasional high peaks could be signaling an emergence of a bug or overhead. In this case an analysis is essential to rectify such occasional peaks. Other than that from this graph an ideal Context Switch for a Multicore Architecture system can be seen.

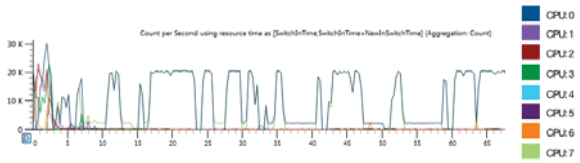

Figure 4.10 - Optimal Context Switch Per Core(8 CPUs)

V. Conclusion and Future Work

In this paper we have successfully implemented a Scalable Hyper Parallel implementation of Canny Edge Detector. We showed that the parallel implementation of CED is highly optimal for Multicore Processors. The results that were observed showed improved CPU usage over wall clock time, efficient CPU usage per core and optimal context switch per core. The results were obtained for both Multicore processor systems with 4 and 8 CPUs. The results showed effective performance of parallel implementation of CED on both Multicore processor systems hence proving that the parallel implementation of CED is scalable. In future we aim to further implement the parallel CED on many core processor systems with 32-64 CPUs to test for validity and robustness of the parallel CED.

References

[1] R. Buchty, V. Heuveline, W. Karl, and J.-P. Weiss, "A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 7, pp. 663–675, 2012.

[2] M. McCool, J. Reinders, and A. Robison, *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.

[3] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012.

[4] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.

[5] P. Bao, D. Zhang, and X. Wu, "Canny edge detection enhancement by scale multiplication," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 9, pp. 1485–1490, Sept 2005.

[6] M. Ali and D. Clausi, "Using the canny edge detector for feature extraction and enhancement of remote sensing images," in *Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International*, vol. 5, 2001, pp. 2298–2300 vol.5.

[7] M. Qi, G. Sun, and G. Chen, "Parallel and simd optimization of image feature extraction," *Procedia Computer Science*, vol. 4, pp. 489–498, 2011.

[8] C. Lin and S. Zhong, "A parallel method for stego image feature extraction on multicore cpu," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, vol. 1. IEEE, 2013, pp. 536–540.

[9] M. Yang, K. Kpalma, J. Ronsin *et al.*, "A survey of shape feature extraction techniques," *Pattern recognition*, pp. 43–90, 2008.

[10] G. Kornaros, "A soft multi-core architecture for edge detection and data analysis of microarray images," *Journal of Systems Architecture*, vol. 56, no. 1, pp. 48–62, 2010.

[11] H. Feng, E. Li, Y. Chen, and Y. Zhang, "Parallelization and characterization of sift on multi-core systems," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, Sept 2008, pp. 14–23.

[12] Y. Luo and R. Duraiswami, "Canny edge detection on nvidia cuda," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 2008, pp. 1–8.
[13] Q. Zhang, Y. Chen, Y. Zhang, and Y. Xu, "Sift implementation and optimization for multi-core systems," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–8.
[14] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on gpus," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 91–104, Jan 2011.
[15] J. Clemons, A. Jones, R. Perricone, S. Savarese, and T. Austin, "Effex: An embedded processor for computer vision based feature extraction," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, June 2011, pp. 1020–1025.
[16] A. D. Robison, "Composable parallel patterns with intel cilk plus," *Computing in Science & Engineering*, vol. 15, no. 2, pp. 0066–71, 2013.
[17] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1369–1386, 2012.