Computing in Parallel

by

Midshipman 1/C Nicholas Vandal

A Senior Project Report Submitted to the Faculty of
the Electrical Engineering Department
United States Naval Academy, Annapolis, Maryland

Faculty Supervisor(s): Dr. Rakvic, Dr. Ngo, LT Blair

April 2009

TABLE OF CONTENTS

FIGURES

TABLES

Abstract

The objective of this project is to explore the gains in performance possible from computing in parallel through different means and analyze the costs-versus-benefit tradeoffs associated with each method. The general technique to accomplish this was to begin initially with a simple sequential, single-threaded application. A multi-threaded version of this application was ported to a high powered, multi-core computer platform, and finally ported to the Cell BE processor within a PlayStation3 running Linux. Execution time is used to develop an understanding of the code acceleration each parallel processing platform is capable of achieving.

The simple application used for benchmarking and initial parallelization, is edge detection on image files of various sizes. Edge detection is implemented by means of simple 2D discrete convolution with kernels of various sizes. OpenCV, a open source computer vision library developed by Intel, is used in all of the implementations. Image processing is a good candidate for parallelization, because it is possible to separate a given frame into multiple sections that can each be independently processed as a separate thread. Additionally, the OpenCV library has already been ported and optimized for the Cell architecture.

The results of this project included: an increased understanding of the complexities involved in writing parallel code on the part of the author, an new Cell-development platform for the Electrical Engineering department at the US Naval Academy, and the conclusion that the Cell BE architecture definitely holds a great deal of promise.

1. Problem Statement

1.1 Need Statement

There will always be an increasing need for raw computational power—scientists, engineers, and even video game programmers will always find a way to eat up more cycles with computationally intensive programs. However, in recent year microprocessor manufacturers have moved away from the paradigm of simply increasing processor clock speed and decreasing feature size—historically the most effective means of increasing CPU capability—and have begun employing multi-core architectures and other means of concurrency. Executing programs in parallel on hardware specifically designed with parallel capabilities appears to be the new model for continuing to increase processor capabilities while not entering into the realm of absurd cooling and power requirements.

However, from the computer scientist's prospective, concurrency in programs is difficult to implement. Historically programmers have though in sequential terms, but in order to harness the capabilities and potential of parallel hardware, it is necessary think concurrently. Often times, this involves completely redesigning an existing program from the ground up and implementing complex synchronization protocols.

1.2 Objective Statement

The objective of this project is to explore the gains in performance possible from computing in parallel through different means and analyze the costs-versus-benefit tradeoffs associated with each method. The general technique to accomplish this will be to begin initially with a simple sequential, single-threaded application and implement it on a single-core PC. A multi-threaded version of this application will then be ported to a multi-core computer platform, implemented in hardware on a FPGA using VHDL, and finally ported to the Cell Processor. As the project progresses, metrics of execution time will be recorded and used to develop an understanding of the code acceleration each parallel processing platform is capable of achieving.

The "simple application" that I intend to utilize for benchmarking and initial parallelization, is real-time image processing to detect edges on video streams captured from a webcam. This is a fairly well established problem with known algorithms that provide adequate solutions. I intend to use the OpenCV open source computer vision library developed by Intel to speed the development of a single threaded version of an edge detection application. Image processing is a good candidate for parallelization, because it is possible to separate a given frame into multiple sections that can each be independently processed as a separate thread. Additionally, the OpenCV library has already been ported and optimized for the Cell architecture.

1.3 Background and Related Work

Parallel programming is based on the simple idea of division of labor—that large problems can be broken up into smaller ones that can be worked on simultaneously. Creating code that executes in parallel from sequential code is not an easy task. The human mind finds it

difficult to grasp concurrently executing code and traditionally computer scientists have dealt only with sequential algorithms, thinking in terms of executing only on instruction at a time. Thinking in parallel is hard because introducing concurrency allows for various new types of software bugs and requires a great deal of work to manage communication and synchronization between simultaneously executing tasks. Ideally, computer programmers long for a compiler that is capable of automatically optimizing sequential code; however, despite a great deal of research, this has not occurred. [1] Producing parallel code typically requires a programmer to isolate a section of code that can be parallelized and figure out an efficient means of doing so. So why write parallel code?

For most of the history of computing, the amazing gains in performance we have experienced were due to two factors: decreasing feature size and increasing clock speed. However, there are fundamental physical limits to this approach—decreasing feature size gets more and more expensive and difficult due to the physics of the photolithographic process used to make CPUs and increasing clock speed results in a subsequent increase in power consumption and heat dissipation requirements. Parallel computation has been in use for many years in high performance computing, however in recent years, multi-core architectures have become the dominate computer architecture for achieving performance gains. The signal that this shift away from ever increasing clock speeds occurred was in May of 2004 when Intel cancelled development of its new single core processors to focus development on dual core technology.[2]

While Intel developed its x86 based multicore designs, a joint effort was already underway by Sony Computer Entertainment, Toshiba Corporation, and IBM began in 2000, with the goal of designing a processor with performance an order of magnitude over that of desktop systems shipping in 2005. The result was the first-generation Cell Broadband Engine (BE) processor, which is a multi-core chip comprised of a 64-bit Power Architecture processor core and eight synergistic processor cores. A high-speed memory controller and high-bandwidth bus interface are also integrated on-chip.[3]

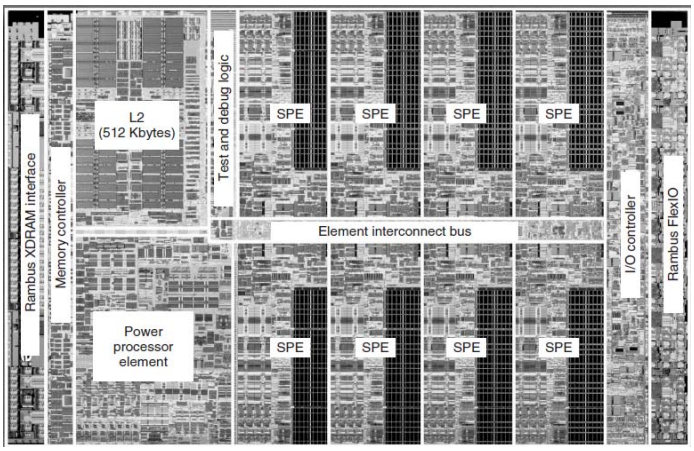


Figure 1: Cell BE Die Photo

The Cell processor has a unique heterogeneous architecture compared to the homogeneous Intel Core architecture. It has a main processor called the Power Processing Element (PPE) (a two-way SMT PowerPC based processor), and eight fully-functional co-processors called the Synergistic Processing Elements, or SPEs. The PPE directs the SPEs where

the bulk of the computation occurs. The PPE is intended primarily for control processing, running operating systems, managing system resources, and managing SPE threads. The SPEs are single-instruction, multiple-data (SIMD) processors with a RISC core. [4]

According to IBM, the Cell BE is capable of achieving in many cases, 10 times the performance of the latest PC processors.[5] The first major commercial application of the Cell processor was in Sony's PlayStation3 game system. The PlayStation3 has only 6 SPU cores available due to one core being reserved by the OS and 1 core being disabled in order to increase production yields. Sony has made it very easy to install a new Linux-based operating system onto the PlayStation3, thereby making the game system a popular choice for experimenting with the Cell BE.

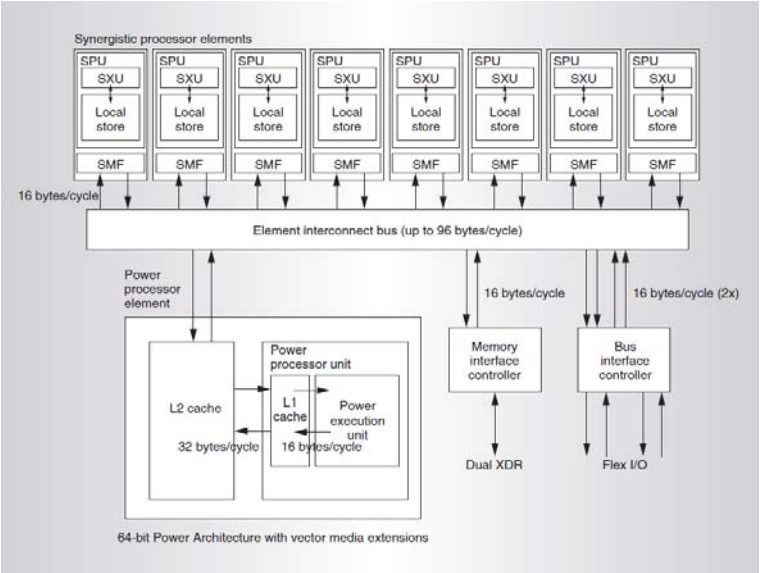


Figure 2: Cell BE High Level Architecture Diagram

In comparing performance of various parallel implementations of an algorithm, several definitions and equations are vital. The term speedup, refers to how much faster a parallel program executes compared to its sequential version. Speedup is calculated based on the following formula:

$$S_p = \frac{T_1}{T_p} \qquad (1)$$

Where p is the number of processors, T_1 is the execution time of the sequential algorithm, and T_p is the execution time of the parallel algorithm run on p processors. Linear speed up is when $S_p=p$. When a speedup of more than N is observed, this is known as super linear speedup.

Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the problem size remains the same when parallelized.

$$S_{Max} = \frac{1}{(1-P) + \frac{P}{N}} \qquad (2)$$

Where P is the proportion of the program that can be parallelized, *(1-P)* is the fraction of the algorithm that is not parallelizable, and N is the number of processors used. The limit as N tends to an infinite number of processors is $\frac{1}{1-P}$. See Figure 1 for an example of Amdahl's law in action for various ratios of P.

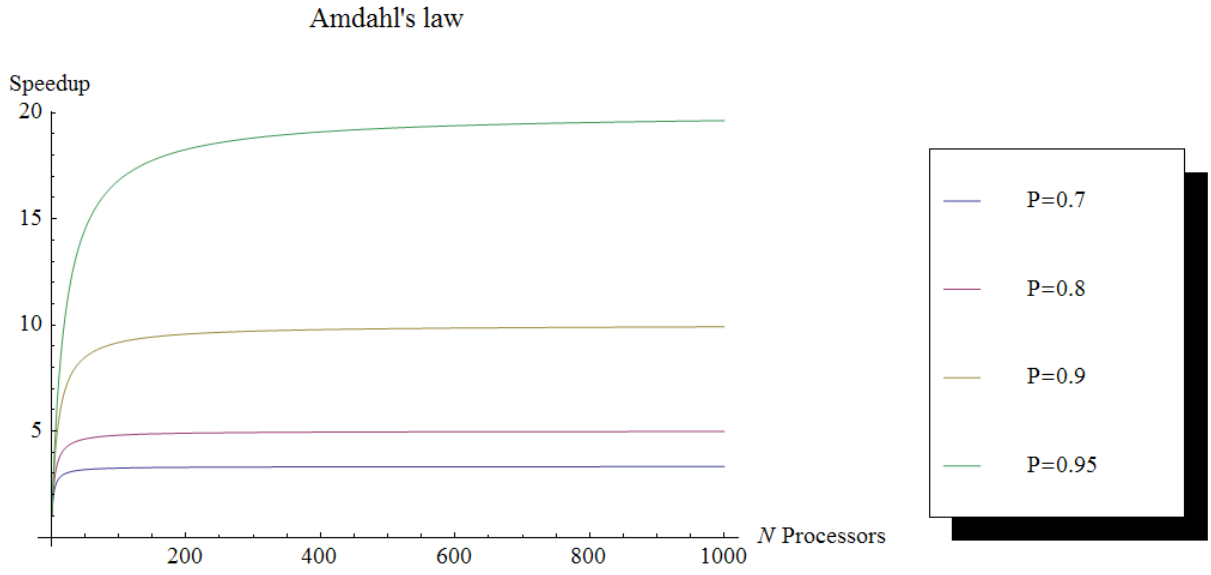


Figure 3: Amdahl's Law in Action for Various Values of P

An embarrassingly parallel problem is a class of problems that require little to no effort to separate the problem into parallel subtasks. These problems typically result where there is no dependency or communication between subtasks. The above 2D convolution filtering problem that forms the core of this project's intended "simple application" is an example of an embarrassingly parallel problem. The individual pixels can each be processed separately. The method of dividing up the 2D convolution algorithm into smaller sub problems will simply be to spatially divide the large input image amongst the available processors. The subtask is identical to the initial task but with a smaller sub image.

2. Requirements Specifications

2.1 The Requirements

Marketing Requirements:

1. The parallel system should produce a significant speedup when compared to a single thread x86 implementation.
2. The system should have low power consumption.
3. The system should be low cost.
4. The system should have good scalability.
5. The system should be easily adaptable to other algorithms.
6. The system should be easy to operate.
7. The system should have a high degree of reliability

Marketing Requirements	Engineering Requirements	Justification
1,4,5	1. The system must achieve an average speedup factor of 2.0 compared to the original single-threaded implementation.	The Cell processor has 8 co-processors in addition to a two-way SMT PowerPC core. The PS3 allows 6 SPE's to be access by applications. According to Amdahl's law, if P is the proportion of a program that can be made parallel, then the maximum speedup that can be achieved by using N processors is $\frac{1}{(1-P)+\frac{P}{N}}$.
5,6	2. The system must use standard libraries, be programmable in C/C++ using GNU compiler/make system.	The C/C++ are widely used to develop performance critical, application code. C/C++ are modern, popular languages with a rich feature set and a great deal of support.
3	3. Total System should cost less than $500.00	Average cost for a Playstation3 (which contains a Cell BE processor) now is approximately $375.00. Additional price allows for additional peripherals and required software.
4	4. System should be able to operate in standalone (single machine) mode, or operate over a high speed LAN.	Allows for scalability if multiple Cell BE processors can be linked together over a LAN to accomplish single common task. PS3 cluster.
2	5. System should be able to operate from standard AC power mains on less than 400 watts.	Comparable to typical power requirements on a standard PC.
7	6. System should be able to operate continuously for over 168 hours.	Long calculations can take months or longer.

Table 1: Marketing Requirements to Engineering Requirements Mapping

		Speed up factor	C/C++ & StdLibs	Cost	Networked	Power	Run time
		+	+	-	+	-	+
1. Speedup	-	↑↑		↓↓	↑	↓↓	
2. Power	-			↓		↑↑	
3. Cost	-			↑↑			
4. Scalability	+	↑↑		↓	↑↑	↓	↓↓
5. Adaptable	+		↑		↑	↓	
6. Easy to operate	+	↓	↑↑				
7. Reliability	+			↓	↓		↑↑

Table 2: Engineering-Marketing Tradeoff Matrix

		Speed up factor	C/C++ & StdLibs	Cost	Networked	Power	Run time
		+	+	-	+	-	+
Speed up factor	+	X		↓	↑		
C/C++ & StdLibs	+	X	X		↑		↑
Cost	-	X	X	X	↓	↓	↓
Networked	+	X	X	X	X	↓	↓
Power	-	X	X	X	X	X	
Run time	+	X	X	X	X	X	X

Table 3: Engineering Tradeoff Matrix

2.2. Constraints

2.2.1. Economic

The price of the entire project should not exceed $400 dollars.

2.2.2 Enviromental

The power consumption of the project should be kept minimal as the system will likely run continuously performing computation.

2.2.3 Manufacturability

The system must be easily manufactured and/or purchased.

2.2.4 Health and Safety

The system should not be able to injure users or anyone around it.

2.2.5 Social

The system should be usable to anyone familiar with Linux/Unix based operating systems.

2.2.6 Political

The system should not require any dubious/illegal software that would reflect poorly on the Navy and the Naval Academy.

2.3 Standards

2.3.1 Testing

The various parallel implementations of the edge detection algorithm must be thoroughly tested for performance on a wide variety of inputs that are representative of real world images. The testing must take into account changing image size and different kernel sizes. Timing of sections of code must be performed consistently for accurate conclusions to be drawn from data.

2.3.2 Documentation

All design and experimentation phases should be properly documented. A thorough step-by-step procedural manual for installing Linux, the IBM SDK, and OpenCV on the PlayStation3 should be generated for future use.

2.3.3 Code

Programming should be performed in C/C++ and use proper formatting, variable names, and comments in order to facilitate future students building on to this work.

3. Design

3.1. Design Alternatives

3.1.1. PIC

- Factors
  - – Limited memory: insufficient size to store large integers
  - – 8-bit architecture: inefficient to process large integers
  - – Single core, not multithreaded
  - – Very slow max clock rate
  - + Easy to understand architecture
- Assessment: The PIC processor is not a viable choice due to its lack of processing capability, memory, and speed. Completely unsuitable for rapid integer factorization.

3.1.2. Single Core Desktop: Non-threaded

- Factors
  - + Flexibility: able to easily implement different algorithms.
  - – Inherently not parallel: sequential
  - + Simplification of algorithm logic: no synchronization required
  - + Familiarity with x86 processor and development environment
- Assessment: Implementation on a single-core x86 desktop computer is the baseline of performance on which the algorithm will be initially implemented and debugged. No speed up because it is not parallel at all.

3.1.3. Single Core Desktop: Multi-threaded

- Factors
  - + Flexibility: able to easily implement different algorithms.
  - – Inherently not parallel on a hardware level. Context switch penalty.
  - + Simplification of algorithm: no message passing between multiple processors.
  - + Minor speed up due to being able to search multiple polynomials
  - + Familiarity with x86 processor and development environment
  - – Increased complexity of algorithm due to threading
- Assessment: Implementation on a single-core x86 desktop computer with multi-threads allows for development of basic synchronization techniques without the added complexity of coordination and job allocation between multiple processors.

3.1.4. Multi-Core “Beast” Computer

- Factors

        - \+ Flexibility: able to easily implement different algorithms.
        - \+ Hardware parallelism due to multiple processor cores
        - \+ Relatively minor change between multi-threaded implementation on single core and the multi-core implementation.
        - \+ Potential for large speedup
        - – Increased complexity of algorithm due to threading
        - \+ Familiarity with x86 processor and development environment
    - Assessment: Implementation on a single-core x86 desktop computer with multi-threads allows for development of basic synchronization techniques without the added complexity of coordination and job allocation between multiple processors.

3.1.5. Cell BE Processor on PS3

- Factors
    - \+ Potential for large speedup due to multi-core hybrid architecture
    - – Increased algorithm complexity due to the need to allocate jobs to Stream Processing Units (SPU's). Optimization on Cell architecture often requires a redesign of algorithm.
    - – Limited SPU memory
    - \+ Potentially able to take advantage of Cell vector processing capability
    - – Limited familiarity with the Cell development environment
- Assessment: The Cell processor represents a viable choice for parallelization of edge detection algorithm but may be difficult to implement due to having to redesign algorithm and learn a unfamiliar architecture.

3.1.6. FPGA

- Factors
    - \+ Inherently parallel; able to potentially parallelize algorithm to extremely high degree.
    - – Programmed in VHDL, not C which is a language I am much more proficient in.
    - \+ Highly specially hardware can be developed and later transferred to an ASIC for extremely high speed processing
    - – Increased design complexity due to inherent concurrency.
- Assessment: The FPGA represents a viable choice for parallelization of the edge detection algorithm but may be difficult to implement due to having to design hardware from scratch. Potentially for extreme speed up, but requires total redesign of custom hardware for different applications.

3.2. Sequential and Cell Algorithm

3.2.1. Level 0

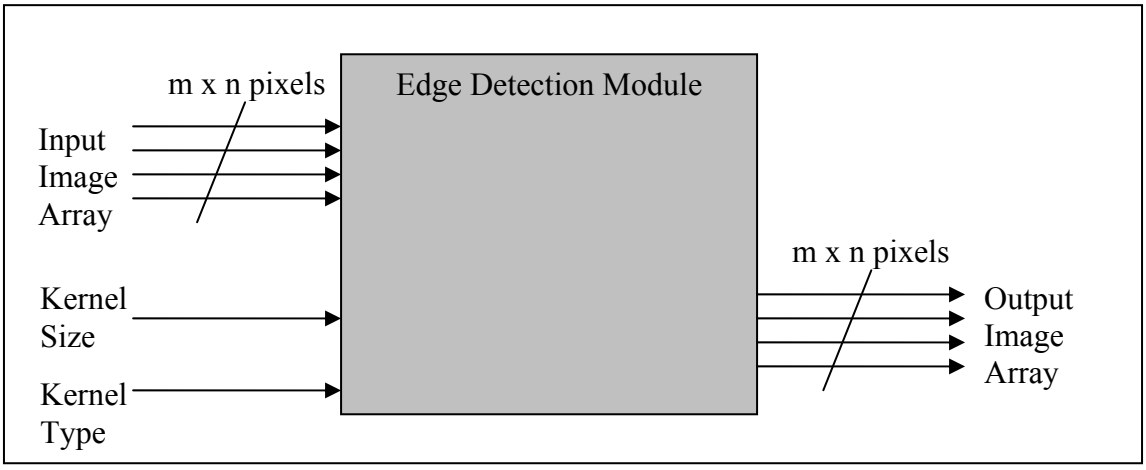


Figure 4: Level 0 Diagram

Module	Edge Detection Module
Inputs	-Input Image Array: m x n pixels loaded from file -Kernel Size: parameter with values of {3, 5, 7, 9}. Sets size of filter image is convolved with. -Kernel Type: parameter used to used type of filter to used.
Outputs	-Output Image Array: m x n pixel binary image with edges highlighted from original image
Functionality	Outputs a binary image with edges highlighted

The sequential version of this code is essentially the same code that runs on the Cell BE in the PlayStation3. Algorithm acceleration is accomplished via the use of OpenCV for the Cell (OpenCV FTC). OpenCV FTC is a version of OpenCV that is specifically optimized for the Cell BE processor found on the PlayStation3. After the library is installed to the Linux operating system running on the Playstation3, setting the environment variable CVCELL_SPENUM to values 1 through 6, the appropriate number of SPU's can be harnessed to work on the algorithm. cvFilter2D is a function that has been specifically optimized to use all 6 SPU's and the SIMD capabilities of the SPU's.

The only restriction placed on cvFilter2D when using the OpenCV FTC is that the srcImg, dstImg, and kernelMat must all be aligned to 16 bytes. This is accomplished within getAlignedKernel.

3.2.2. Level 1

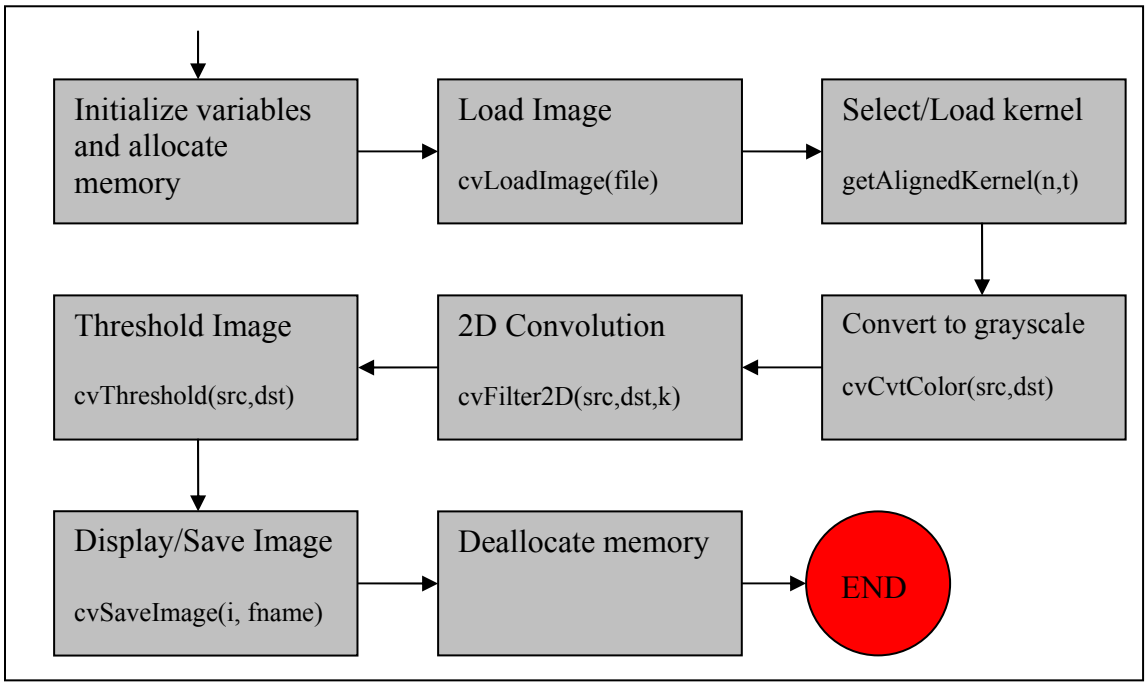


Figure 5: Level 1 Layout -- Sequential

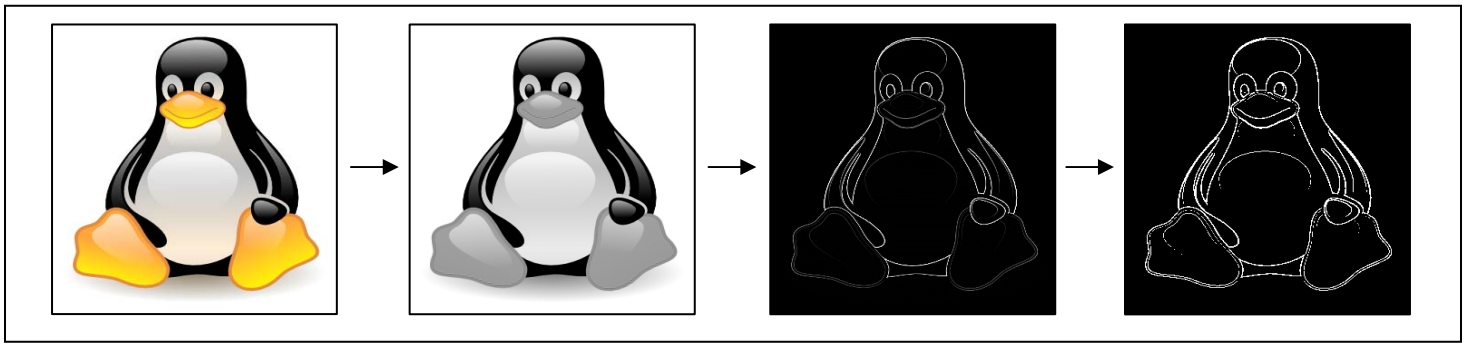


Figure 6: Edge Detection Steps

Module	Initialize
Inputs	None
Outputs	None
Functionality	Allocates dynamic memory for storage of srcImg, grayImg, outImg, kernel

Module	Load Image
Inputs	-filename: name of image to load -img: IplImage array to fill
Outputs	-img: IplImage array representing image to perform edge

	detection on.
Functionality	Outputs array representing image to perform edge detection on when given filename

Module	**Select/Load kernel**
Inputs	-size: size of the kernel/filter to use {3,5,7,9} -type: type of image filter {StobelH,StobelV, Prewitt, Laplacian of Gaussian(LoG)}
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired.
Functionality	Outputs a kernel/filter for use in 2D convolution. Constants based on Matlab 'fspecial'.

Module	**Convert to grayscale**
Inputs	-src: source color image -dst: destination grayscale image
Outputs	-dst: grayscale representation of the color input image
Functionality	Outputs a grayscale conversion of the input color image using cvCvtColor(src,dst)

Module	**2D Convolution**
Inputs	-src: source image to perform filtering on -dst: destination image to output result of filtering to -k: kernel to convolve with source image
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired.
Functionality	Performs 2D convolution with cvFilter2D function.

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

Figure 7: Sobel Horizontal and Vertical 3x3 Kernels

Module	**Threshold Image**
Inputs	-src: source image to perform thresholding on -dst: destination image to output result of thresholding to
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired.
Functionality	Performs thresholding with cvThreshold function

Module	**Display/Save Image**
Inputs	-img: image to save to file -filename: filename to save to
Outputs	None
Functionality	Writes image out to disk with cvSaveImage function

3.3. 4 Threaded and 8 Threaded Multi-Core "Beast" Algorithm

3.3.1. Level 0

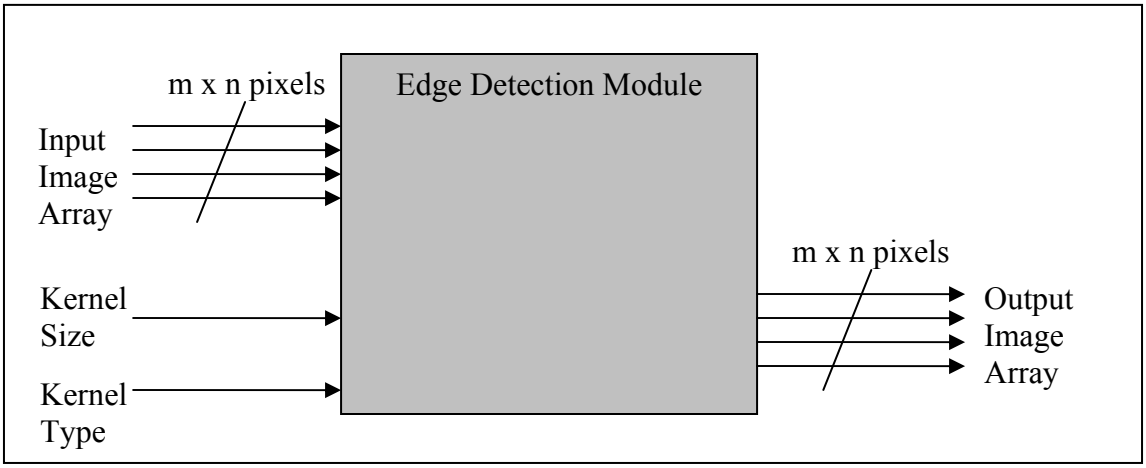


Figure 8: Level 0 Diagram

Module	Edge Detection Module
Inputs	-Input Image Array: m x n pixels loaded from file -Kernel Size: parameter with values of {3, 5, 7, 9}. Sets size of filter image is convolved with. -Kernel Type: parameter used to used type of filter to used.
Outputs	-Output Image Array: m x n pixel binary image with edges highlighted from original image
Functionality	Outputs a binary image with edges highlighted

The 4 and 8 threaded implementations of the edge detection algorithm that run on the Beast achieves acceleration by breaking the image up spatially into either 4 or 8 sub-regions and performs 2D convolution on each region simultaneously with 1 thread assigned to process a particular sub-region.

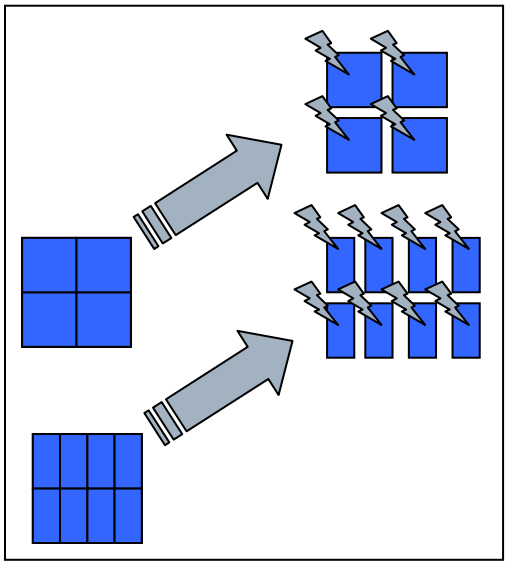


Figure 9: Spatial Division of Labor

3.3.2. Level 1

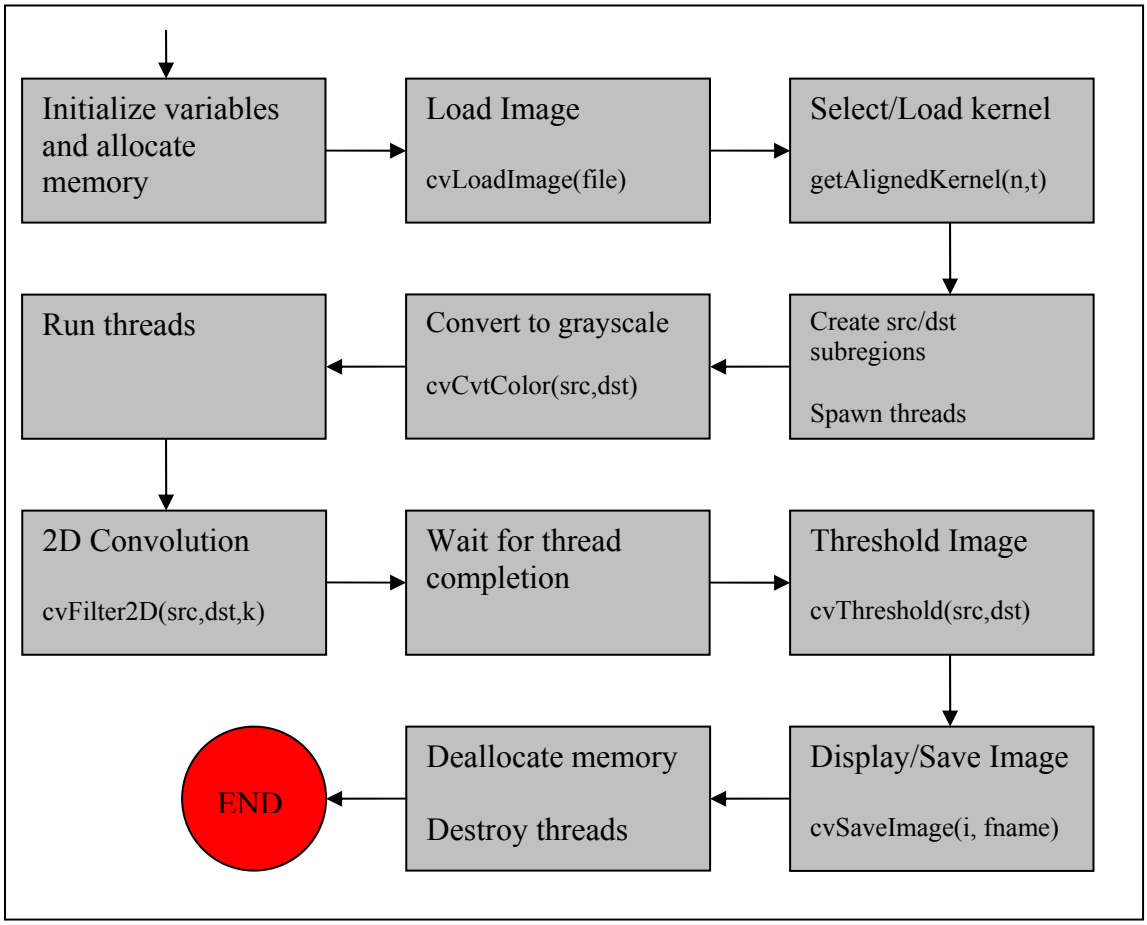


Figure 10: Level 1 Layout – Threaded

Module	**Initialize**
Inputs	None
Outputs	None
Functionality	Allocates dynamic memory for storage of srcImg, grayImg, outImg, kernel

Module	**Load Image**
Inputs	-filename: name of image to load -img: IplImage array to fill
Outputs	-img: IplImage array representing image to perform edge detection on.
Functionality	Outputs array representing image to perform edge detection on when given filename

Module	Select/Load kernel
Inputs	-size: size of the kernel/filter to use {3,5,7,9} -type: type of image filter {StobelH,StobelV, Prewitt, Laplacian of Gaussian(LoG)}
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired.
Functionality	Outputs a kernel/filter for use in 2D convolution. Constants based on Matlab 'fspecial'.

Module	Create subregions and Spawn Threads
Inputs	None
Outputs	Defines subregions and spawns threads to operate on subregions

Functionality	`#define IMAGE_DATA_PTR(origIm, x, y) ((origIm)->imageData + (y)*(origIm)->widthStep + (x)*(origIm)->nChannels)`
	
	`for(int i=0; i<THREADCOUNT; ++i)`
	`{`
	`  //Initialize source and desination`
	`  sub_img_src[i] = cvCreateImageHeader(`
	`  cvSize(img->width/4,`
	`  img->height/2),`
	`  img->depth,`
	`  img->nChannels );`
	
	`  sub_img_dst[i] = cvCreateImageHeader(…);`
	
	`  //Image data`
	`  sub_img_src[i]->imageData = IMAGE_DATA_PTR(img,`
	`  (img->width/4)*(i % 4),`
	`  (img->height/2)*(i / 4));`
	`}`
	
	`.`
	`.`
	`.`
	
	`// Create worker threads`
	`for( int i=0; i < THREADCOUNT; i++ )`
	`{`
	`  //Create worker thread`
	`  aThread[i] = (HANDLE)_beginthreadex( …,`
	`  &helperThreadFunc,…);`
	
	`  //Create event array to signal back to main that thread is`
	`  //ready for more work`
	`  hWorkerThreadDone[i] = CreateEvent(…);`
	
	`  //Create event array to signal to worker threads that they`
	`  //can resume work`
	`  hEventMoreWorkToDo[i] = CreateEvent(…);`
	`}`

Module	Convert to grayscale
Inputs	-src: source color image -dst: destination grayscale image
Outputs	-dst: grayscale representation of the color input image
Functionality	Outputs a grayscale conversion of the input color image using

	cvCvtColor(src,dst)

Module	2D Convolution/Wait for Thread Completion
Inputs	-src: source image to perform filtering on -dst: destination image to output result of filtering to -k: kernel to convolve with source image
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired. Performs 2D convolution with cvFilter2D function on subregion.

Functionality	`// Helper function`
	`unsigned __stdcall helperThreadFunc( void* pArguments )`
	`{`
	`  do{`
	`  //Do  work on specific quadrant based upon argument`
	`  int q = (int)pArguments;`
	`  DWORD dwRet;`
	`  cvFilter2D(sub_img_src[q],sub_img_dst[q],kernel);`
	`  //Signal that has completed work`
	`  SetEvent(hWorkerThreadDone[q]);`
	`  // wait for moreWork event from main thread`
	`  dwRet =`
	`  WaitForSingleObject(hEventMoreWorkToDo[q],INFINITE);`
	`  }while(dwRet == WAIT_OBJECT_0);`
	`}`

Module	Threshold Image
Inputs	-src: source image to perform thresholding on -dst: destination image to output result of thresholding to
Outputs	-kernel: (size x size) matrix of floats filled with appropriate constants for the type of filter desired.
Functionality	Performs thresholding with cvThreshold function

Module	Display/Save Image
Inputs	-img: image to save to file -filename: filename to save to
Outputs	None
Functionality	Writes image out to disk with cvSaveImage function

3.4 Installation of PlayStation3 Linux/OpenCV Enivironment

Fedora Core 8 was chosen for installation onto the PlayStation3. Fedora Core 8 is not the most recent release of Fedora but was chosen because it is the most recent release that has been fully adapted to the PlayStation3. Additionally, the installation procedures available online for FC8 are the most detailed and complete of any Linux distribution. Furthermore, the IBM SDK, which is required for writing code that runs on the Cell's SPUs is specifically only released for the commercial Red Hat Enterprise Edition Linux or the freely available Fedora Core. Installation was quite challenging and required learning a great more about Linux. Complete

instructions for the installation of Fedora Core 8, OpenCV FTC, and the IBM SDK are available in the appendix.

4. Design Verification

4.1. Testing

4.1.1. Test Procedures

Two dimensional convolution is dependant on both image size and kernel size. Because the convolution operation is performed for every pixel in the image regardless of intensity values and with every value in the kernel mask, assuming constant time multiplication operations, execution time depends solely on image size and kernel size, and does not vary with different images of the same size.

Testing was accomplished by performing 2D filtering on each of the images shown below in Figure 10, with each of the algorithm implementations (Sequential on the Beast, 4 Threaded on the Beast, 8 Threaded on the Beast, and the Cell), for each value of k (kernel size = {3x3,5x5,7x7,9x9}). The execution time for the 2D convolution was timed with the internal OpenCV timing functions. This was repeated 100 times to get an average execution time for each implementation on each image with each value of k. Additionally, the for the Celll implementation, average timing was determined with 0, 1, 2, 3, 4, 5, and 6 of the SPU's engaged in the algorithm.

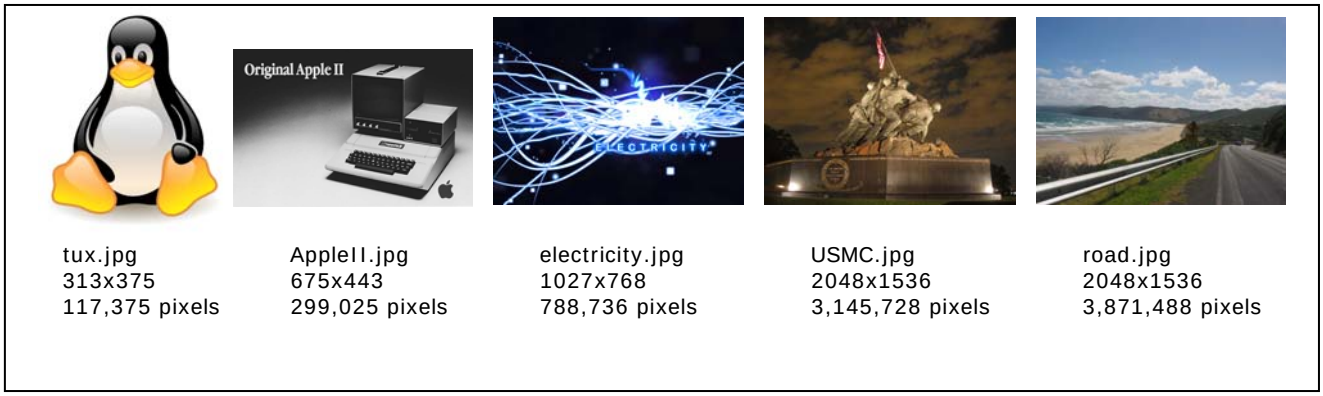


Figure 11: Test Images

4.1.2. Test Results – Raw Data

4.1.2.1 Test Results: The Beast—Sequential Algorithm

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	1.892851	1.376432	1.4075665	1.876633	1.880583	4.036974	7.240141	11.49151
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	4.945519	3.558463	3.736509	4.92891	4.942271	10.40872	18.6034	29.73373
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	12.66212	8.850724	9.28896	12.51254	12.50351	26.75225	48.45543	76.60625
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	57.26867	40.693724	44.161032	56.09959	56.05177	113.4188	198.4658	310.9962
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	63.39137	44.625642	47.65835	63.36377	63.32416	138.7151	239.5162	377.7699

4.1.2.2 Test Results: The Beast—4 Threaded Algorithm

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.459681	0.274883	0.310832	0.397546	0.443957	0.743269	1.395195	2.399931
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.998024	0.924378	0.974663	1.040211	1.003404	1.984335	3.739098	6.41152
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	2.285943	2.25442	2.448525	2.281163	2.281163	2.478455	4.67025	9.624933
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	9.175519	8.194233	8.510738	9.109451	8.513911	19.13789	40.13382	68.25539
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	12.4395	12.05128	11.19894	13.26212	13.04332	22.13321	48.49174	84.11262

4.1.2.3 Test Results: The Beast—8 Threaded Algorithm

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.789577	0.542417	0.45643	0.617631	0.89675	1.03726	1.50238	1.677478
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.903453	0.985101	0.965368	1.354717	0.722438	2.098584	3.612621	4.939245
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	1.870769	1.60076	2.276623	2.709634	2.602848	3.441413	7.336076	12.45618
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	9.109546	5.248073	5.854174	7.452136	6.904256	15.37592	23.30556	35.3209
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	8.712414	8.805561	9.154754	8.797865	8.906652	17.4904	31.31774	47.95944

4.1.2.4 Test Results: PlayStation3 – 0 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	6.16689	4.96359	4.96313	5.97448	6.98942	10.72466	17.83287	27.25142
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	15.65512	12.87714	12.97284	15.43472	15.66752	27.67384	45.77588	69.56842
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	40.42994	33.45551	33.61424	39.80647	40.3982	71.74013	118.53	178.7613
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	167.8645	138.8746	143.2819	166.2492	168.0822	294.0624	481.1291	720.4315
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	198.0736	164.8791	168.5334	198.0625	199.7412	355.3384	586.5384	884.5536

4.1.2.5 Test Results: PlayStation3 – 1 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.97453	0.96429	0.96999	0.96534	0.96548	2.34662	3.94656	6.38535
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	2.27399	2.25852	2.25915	2.25826	2.25777	5.76557	9.79717	15.98192
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	5.79366	5.78273	5.77934	5.78185	5.7822	15.0209	25.63112	41.94703
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	22.58162	22.56469	22.57662	22.56617	22.55909	59.38233	101.7498	167.0727
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	27.68237	27.66209	27.66425	27.66895	27.66734	72.95983	125.1421	205.5385

4.1.2.6 Test Results: PlayStation3 – 2 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.53251	0.51824	0.51756	0.51671	0.51918	1.20876	2.01152	3.21784
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	1.19182	1.17183	1.17432	1.17592	1.17725	2.91821	4.92423	8.02483
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	2.97448	2.9592	2.96604	2.96276	2.968	7.57552	12.87634	21.02344
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	11.51679	11.49844	11.48151	11.50538	11.48015	29.89653	50.95761	83.59116
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	14.12843	14.08733	14.08852	14.08906	14.09663	36.69913	62.66073	102.821

4.1.2.7 Test Results: PlayStation3 – 3 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.4092	0.37611	0.3764	0.3742	0.37492	0.83714	1.36941	2.1827
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.82862	0.81512	0.81804	0.82011	0.8198	1.97943	3.33101	5.42281
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	2.03835	2.02642	2.02645	2.03581	2.03832	5.09543	8.62985	14.05426
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	7.81062	7.79274	7.79422	7.79858	7.79415	20.05273	34.09878	55.76958
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	9.57662	9.54743	9.54802	9.55161	9.67248	24.61599	41.90649	68.59352

4.1.2.8 Test Results: PlayStation3 – 4 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.32103	0.3043	0.30514	0.30729	0.30681	0.64905	1.04522	1.67016
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.65487	0.64344	0.64633	0.64358	0.65351	1.51627	2.55371	4.14077
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	1.58608	1.5648	1.56758	1.57092	1.5658	3.87583	6.50867	10.58255
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	5.98913	5.96188	5.93953	5.95131	5.94084	15.16739	25.68039	41.89166
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	7.30724	7.28963	7.28118	7.2993	7.28316	18.58406	31.55081	51.52295

4.1.2.9 Test Results: PlayStation3 – 5 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.28821	0.26577	0.2706	0.26757	0.34041	0.54567	0.86667	1.35335
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.56421	0.54128	0.54393	0.54403	0.55946	1.23601	2.08232	3.34803
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	1.29943	1.28514	1.28416	1.28232	1.28306	3.13485	5.31785	8.60567
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	4.87802	4.83309	4.85669	4.85569	4.84681	12.20946	20.61728	33.68608
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	5.97154	5.95794	5.95796	5.96185	5.96926	15.11475	25.5747	41.68633

4.1.2.10 Test Results: PlayStation3 – 6 SPUs

			Original	Horizontal	Vertical	P	Laplacian of Gauss (LoG)			
			3x3	*3x3*	*3x3*	*3x3*	*3x3*	*5x5*	*7x7*	*9x9*
"tux.jpg"										
W	*H*	*Num Pixels*								
313	*375*	*117375*	0.26216	0.24546	0.2453	0.24528	0.2459	0.47631	0.74039	1.15902
"AppleII.jpg"										
W	*H*	*Num Pixels*								
675	*443*	*299025*	0.49318	0.47366	0.47702	0.47594	0.47483	1.08334	1.80908	2.8867
"electricity.jpg"										
W	*H*	*Num Pixels*								
1027	*768*	*788736*	1.12607	1.10914	1.11467	1.1139	1.11441	2.65101	4.42662	7.12137
"USMC.jpg"										
W	*H*	*Num Pixels*								
2048	*1536*	*3145728*	4.15295	4.12721	4.10998	4.14333	4.13766	10.26787	17.30628	28.12818
"road.jpg"										
W	*H*	*Num Pixels*								
2272	*1704*	*3871488*	5.06902	5.0463	5.0375	5.03933	5.05152	12.60309	21.26276	34.59322

4.1.3. Test Results – Kernel Size v. Execution Time

Beast
Sequential **Kernel Size**

Image	Image Size	3	5	7	9
"tux.jpg"	117375	1.686813	4.036974	7.240141	11.49151
"AppleII.jpg"	299025	4.422334	10.40872	18.6034	29.73373
"electricity.jpg"	788736	11.16357	26.75225	48.45543	76.60625
"USMC.jpg"	3145728	50.85496	113.4188	198.4658	310.9962
"road.jpg"	3871488	56.47266	138.7151	239.5162	377.7699

Beast 4 Threads

"tux.jpg"	117375	0.37738	0.743269	1.395195	2.399931
"AppleII.jpg"	299025	0.988136	1.984335	3.739098	6.41152
"electricity.jpg"	788736	2.310243	2.478455	4.67025	9.624933
"USMC.jpg"	3145728	8.70077	19.13789	40.13382	68.25539
"road.jpg"	3871488	12.39903	22.13321	48.49174	84.11262

Beast 8 Threads

"tux.jpg"	117375	0.660561	1.03726	1.50238	1.677478
"AppleII.jpg"	299025	0.986215	2.098584	3.612621	4.939245
"electricity.jpg"	788736	2.212127	3.441413	7.336076	12.45618
"USMC.jpg"	3145728	6.913637	15.37592	23.30556	35.3209
"road.jpg"	3871488	8.875449	17.4904	31.31774	47.95944

PS3 – 0 SPU's

"tux.jpg"	117375	5.811502	10.72466	17.83287	27.25142
"AppleII.jpg"	299025	14.52147	27.67384	45.77588	69.56842
"electricity.jpg"	788736	37.54087	71.74013	118.53	178.7613
"USMC.jpg"	3145728	156.8705	294.0624	481.1291	720.4315
"road.jpg"	3871488	185.8579	355.3384	586.5384	884.5536

PS3 – 1 SPU's

"tux.jpg"	117375	0.967926	2.34662	3.94656	6.38535
"AppleII.jpg"	299025	2.261538	5.76557	9.79717	15.98192
"electricity.jpg"	788736	5.783956	15.0209	25.63112	41.94703
"USMC.jpg"	3145728	22.56964	59.38233	101.7498	167.0727
"road.jpg"	3871488	27.669	72.95983	125.1421	205.5385

PS3 – 2 SPU's

Image	Image Size	3	5	7	9
"tux.jpg"	117375	0.52084	1.20876	2.01152	3.21784
"AppleII.jpg"	299025	1.178228	2.91821	4.92423	8.02483
"electricity.jpg"	788736	2.966096	7.57552	12.87634	21.02344
"USMC.jpg"	3145728	11.49645	29.89653	50.95761	83.59116
"road.jpg"	3871488	14.09799	36.69913	62.66073	102.821

PS3 – 3 SPU's

"tux.jpg"	117375	0.382166	0.83714	1.36941	2.1827
"AppleII.jpg"	299025	0.820338	1.97943	3.33101	5.42281
"electricity.jpg"	788736	2.03307	5.09543	8.62985	14.05426
"USMC.jpg"	3145728	7.798062	20.05273	34.09878	55.76958
"road.jpg"	3871488	9.579232	24.61599	41.90649	68.59352

PS3 – 4 SPU's

"tux.jpg"	117375	0.308914	0.64905	1.04522	1.67016
"AppleII.jpg"	299025	0.648346	1.51627	2.55371	4.14077
"electricity.jpg"	788736	1.571036	3.87583	6.50867	10.58255
"USMC.jpg"	3145728	5.956538	15.16739	25.68039	41.89166
"road.jpg"	3871488	7.292102	18.58406	31.55081	51.52295

PS3 – 5 SPU's

"tux.jpg"	117375	0.286512	0.54567	0.86667	1.35335
"AppleII.jpg"	299025	0.550582	1.23601	2.08232	3.34803
"electricity.jpg"	788736	1.286822	3.13485	5.31785	8.60567
"USMC.jpg"	3145728	4.85406	12.20946	20.61728	33.68608
"road.jpg"	3871488	5.96371	15.11475	25.5747	41.68633

PS3 – 6 SPU's

"tux.jpg"	117375	0.24882	0.47631	0.74039	1.15902
"AppleII.jpg"	299025	0.478926	1.08334	1.80908	2.8867
"electricity.jpg"	788736	1.115638	2.65101	4.42662	7.12137
"USMC.jpg"	3145728	4.134226	10.26787	17.30628	28.12818
"road.jpg"	3871488	5.048734	12.60309	21.26276	34.59322

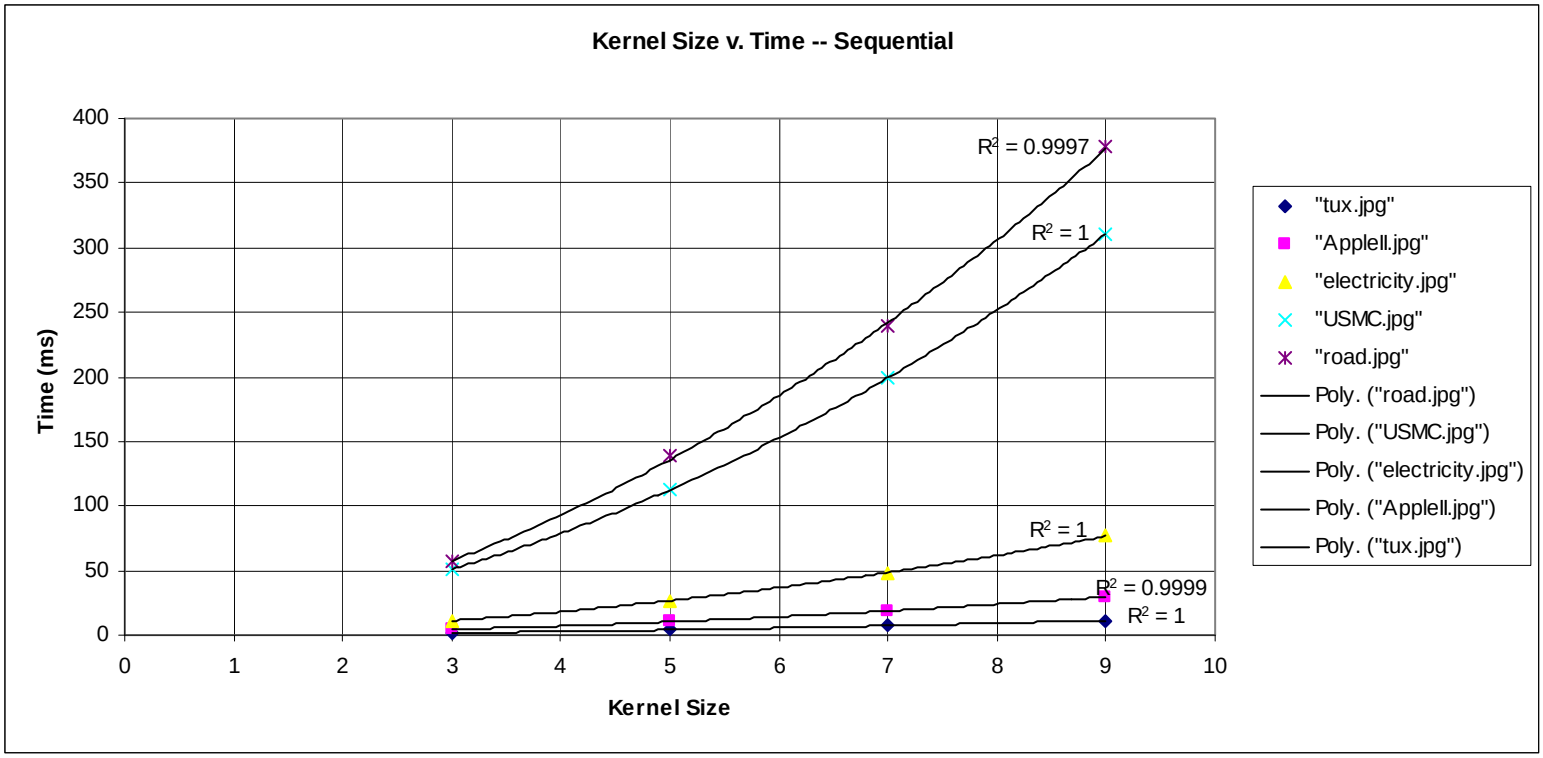


Figure 12: Kernel Size v. Time constant Implementation—Sequential Algorithm

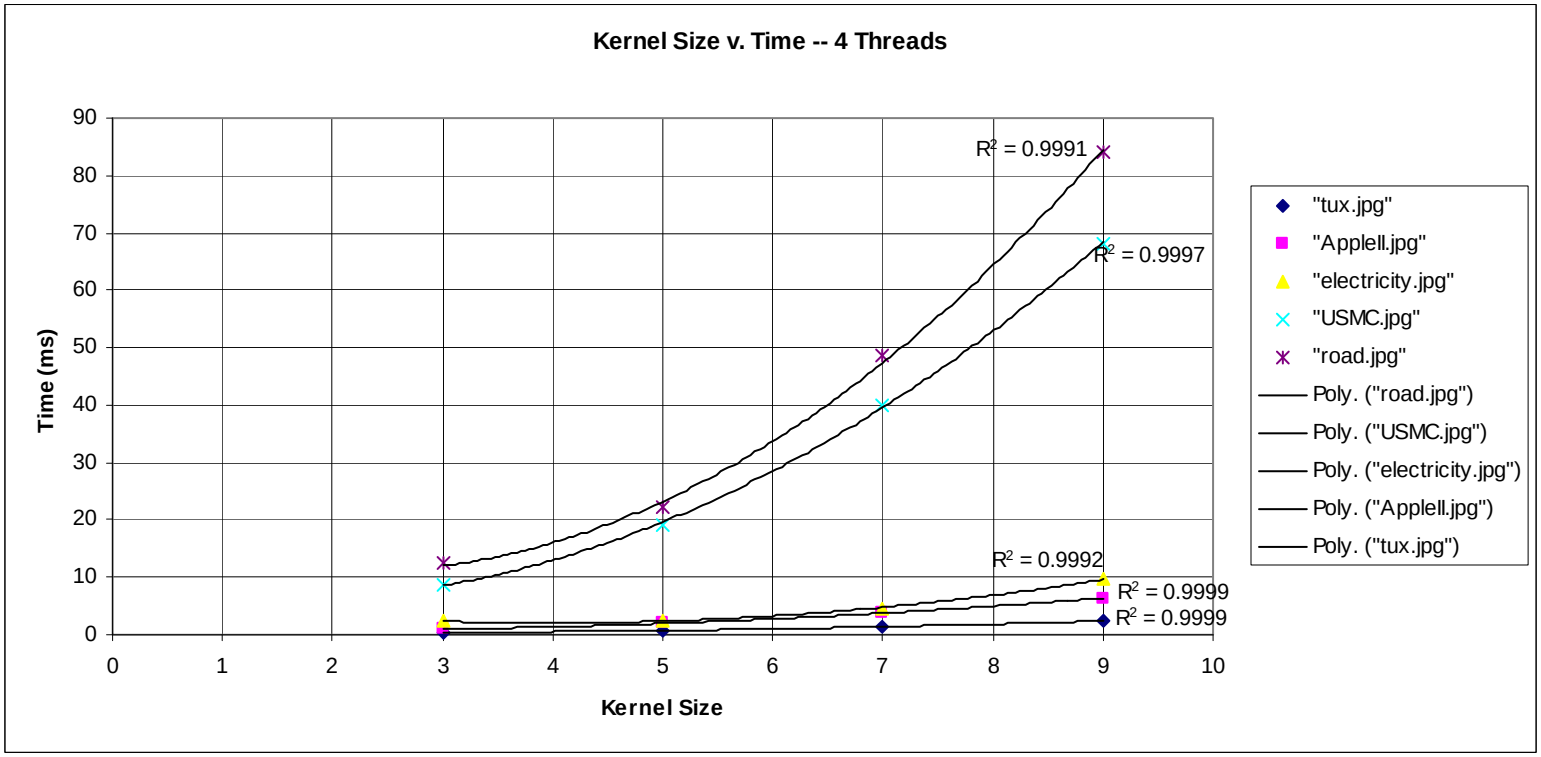


Figure 13: Kernel Size v. Time constant Implementation—4 Threaded Beast

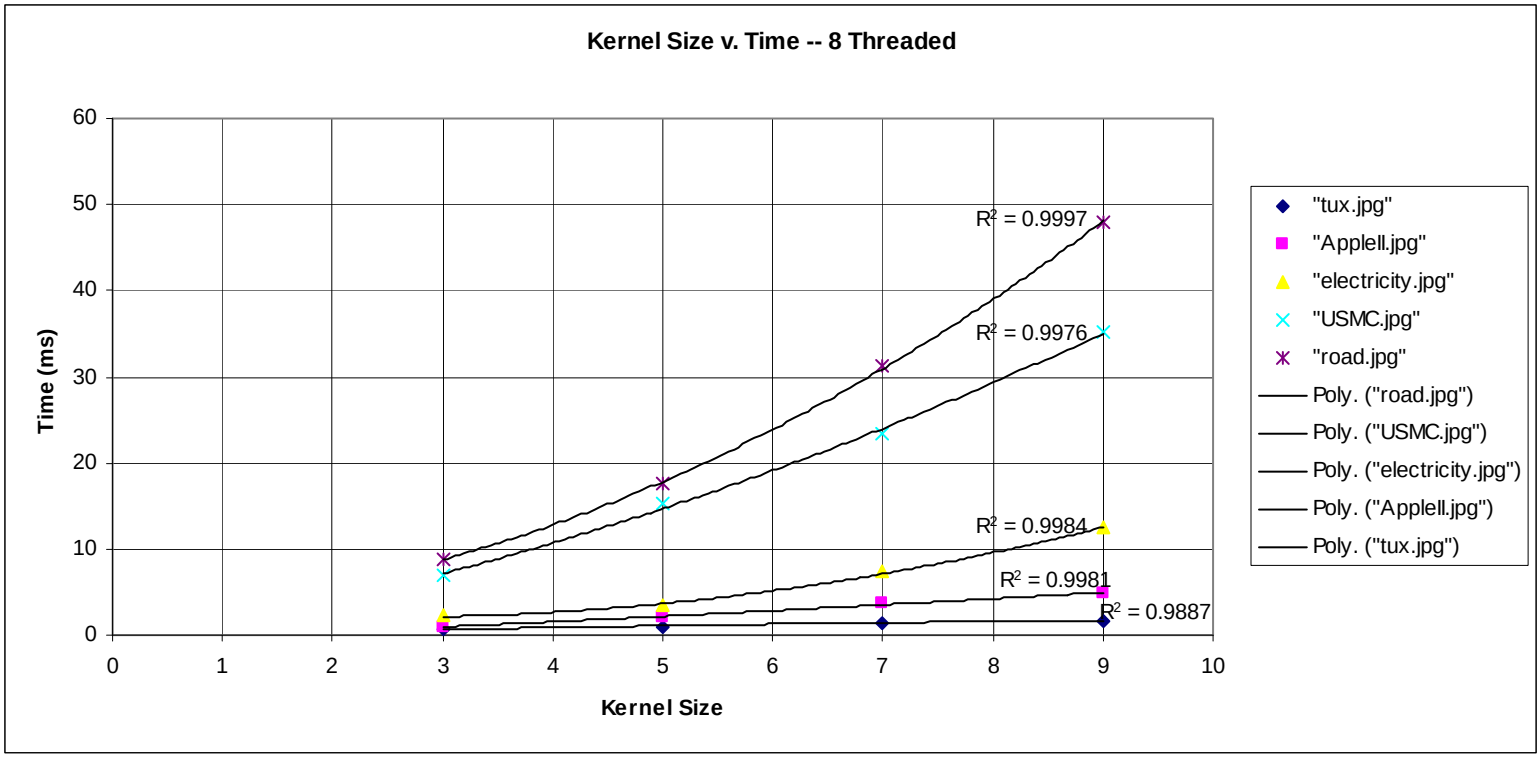


Figure 14: Kernel Size v. Time constant Implementation—8 Threaded Beast

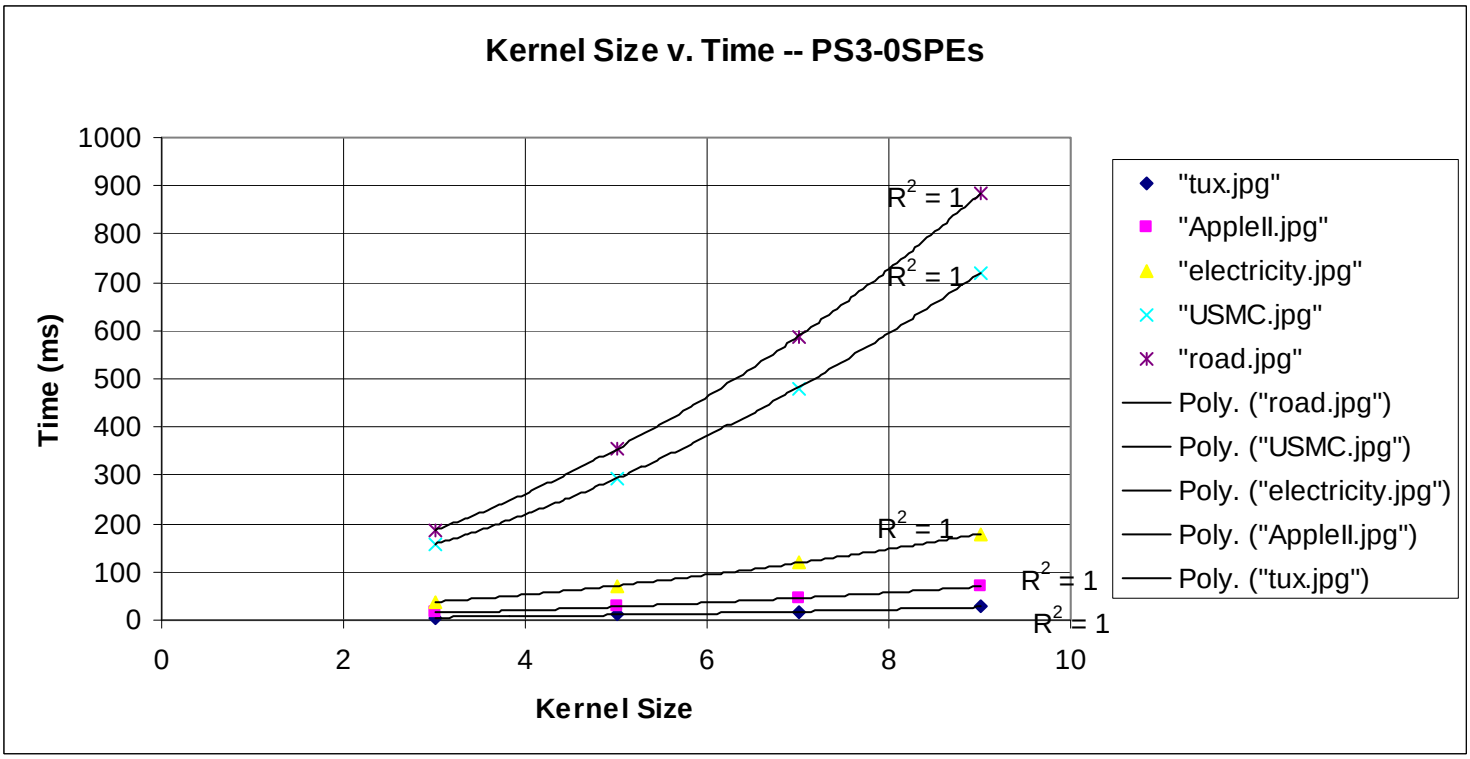


Figure 15: Kernel Size v. Time constant Implementation—PS3-0SPUs

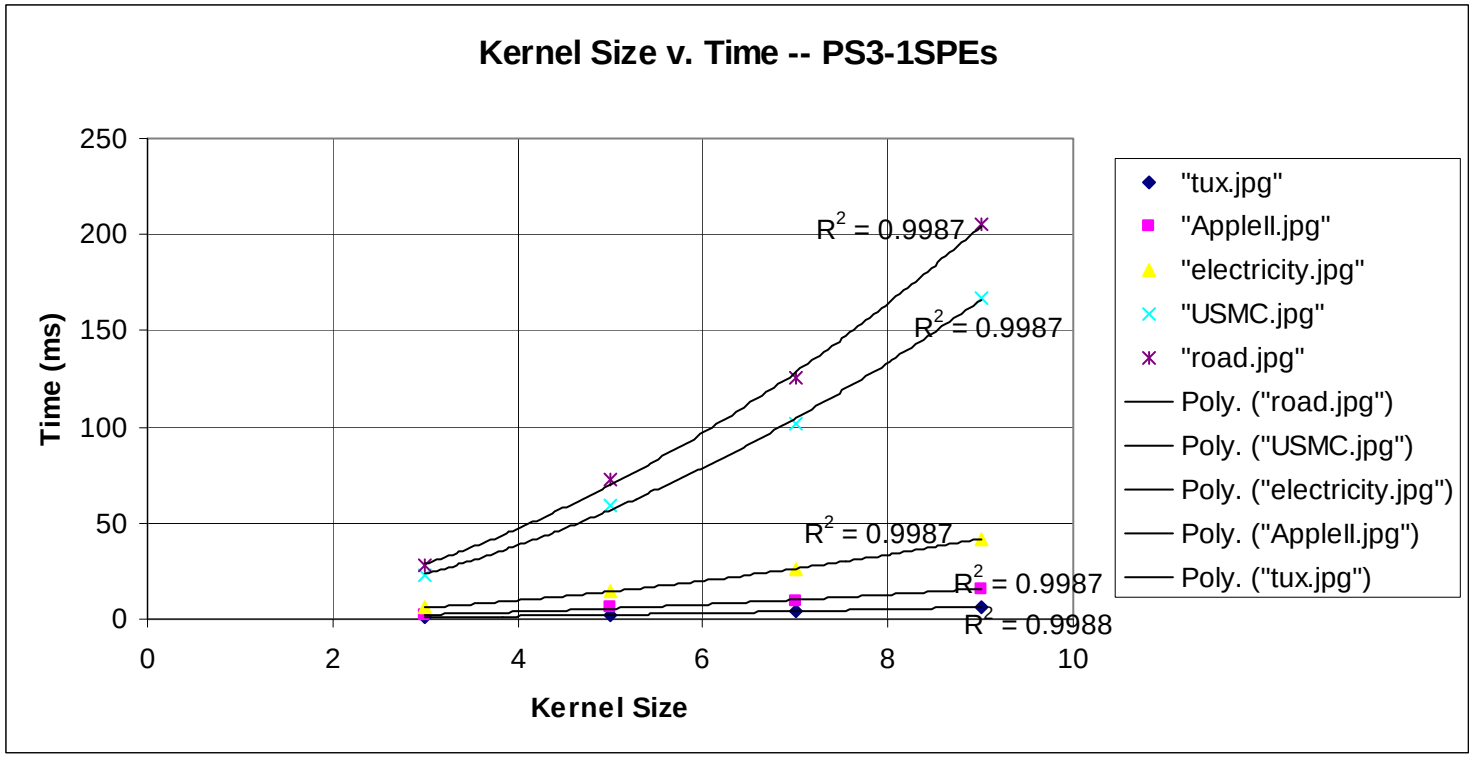


Figure 16: Kernel Size v. Time constant Implementation—PS3-1SPUs

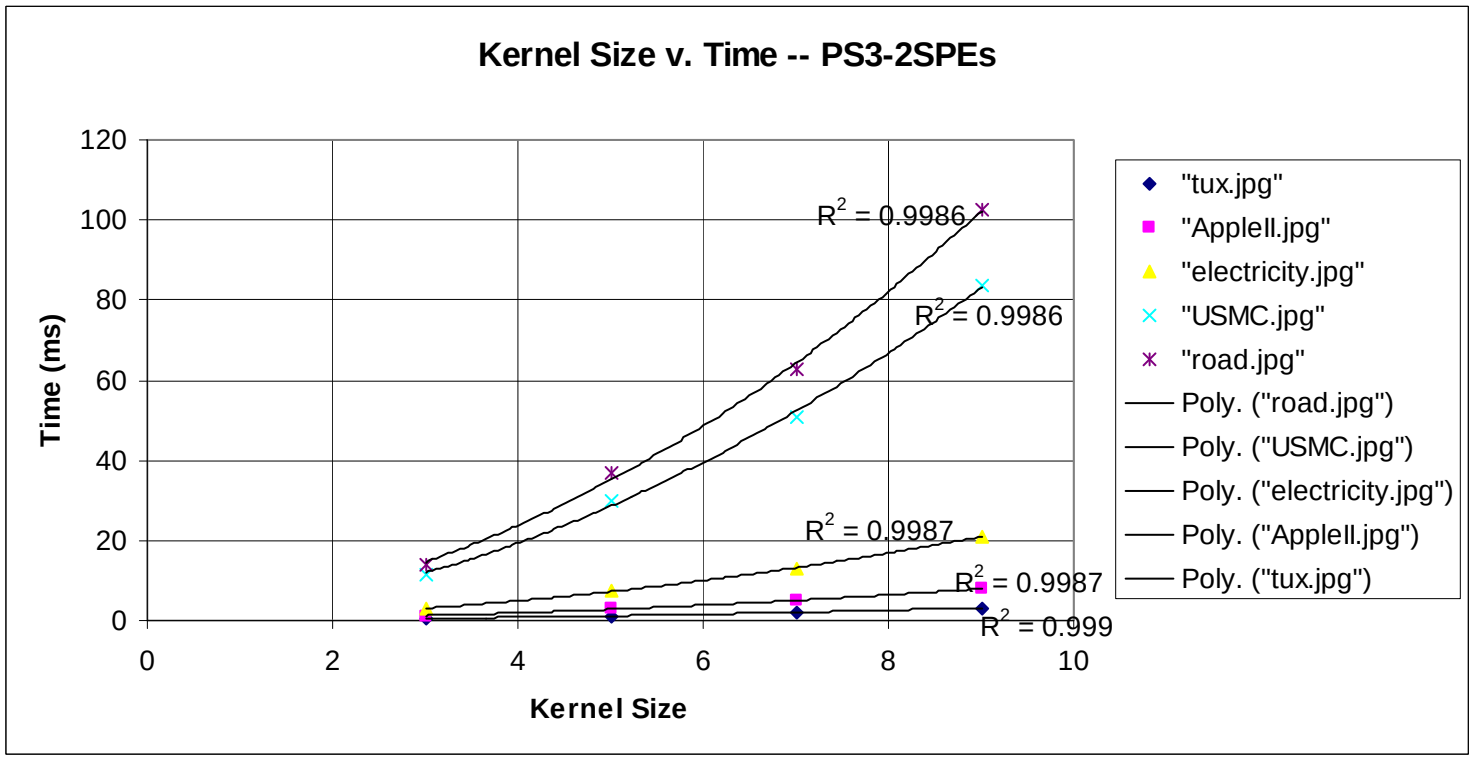


Figure 17: Kernel Size v. Time constant Implementation—PS3-2SPUs

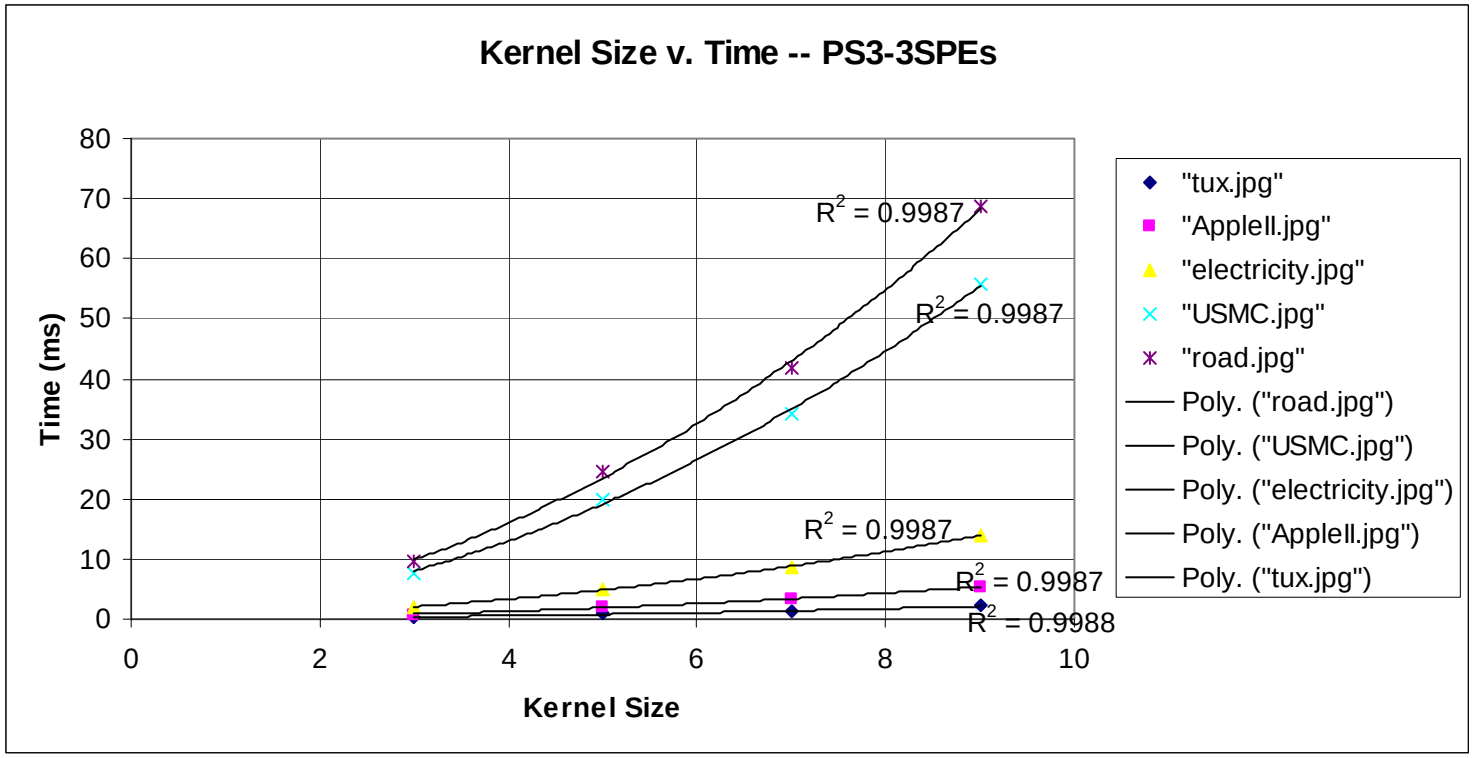


Figure 18: Kernel Size v. Time constant Implementation—PS3-3SPUs

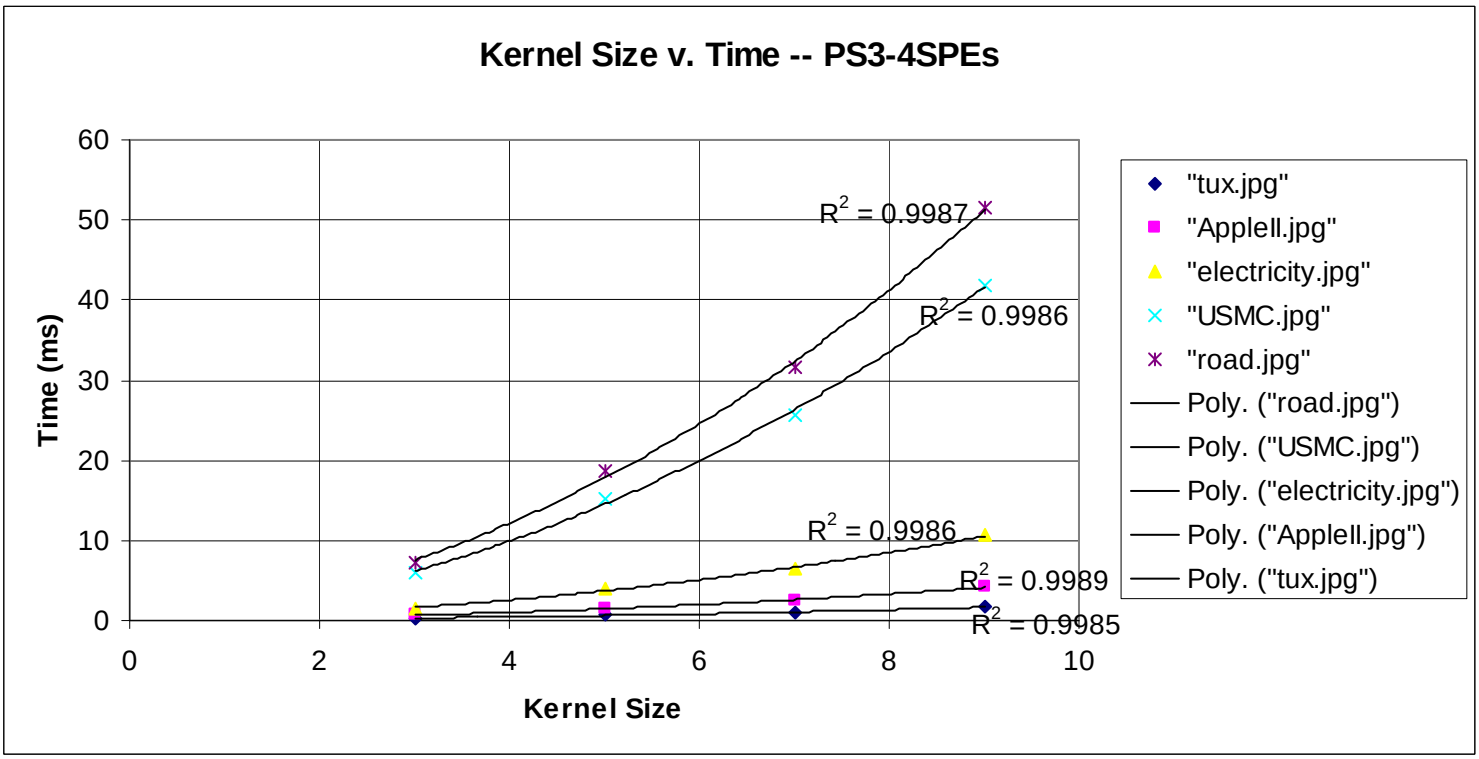


Figure 19: Kernel Size v. Time constant Implementation—PS3-4SPUs

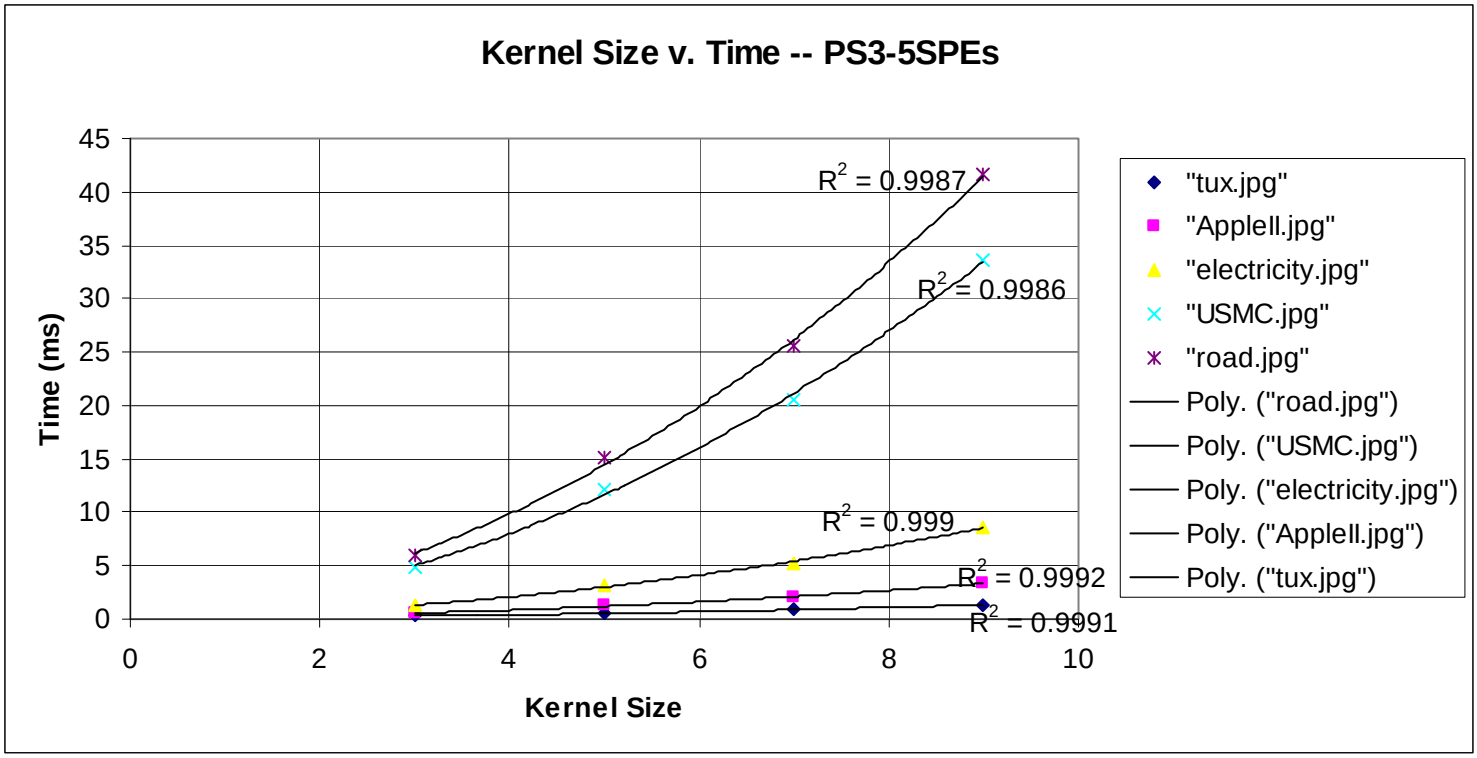


Figure 20: Kernel Size v. Time constant Implementation—PS3-5SPUs

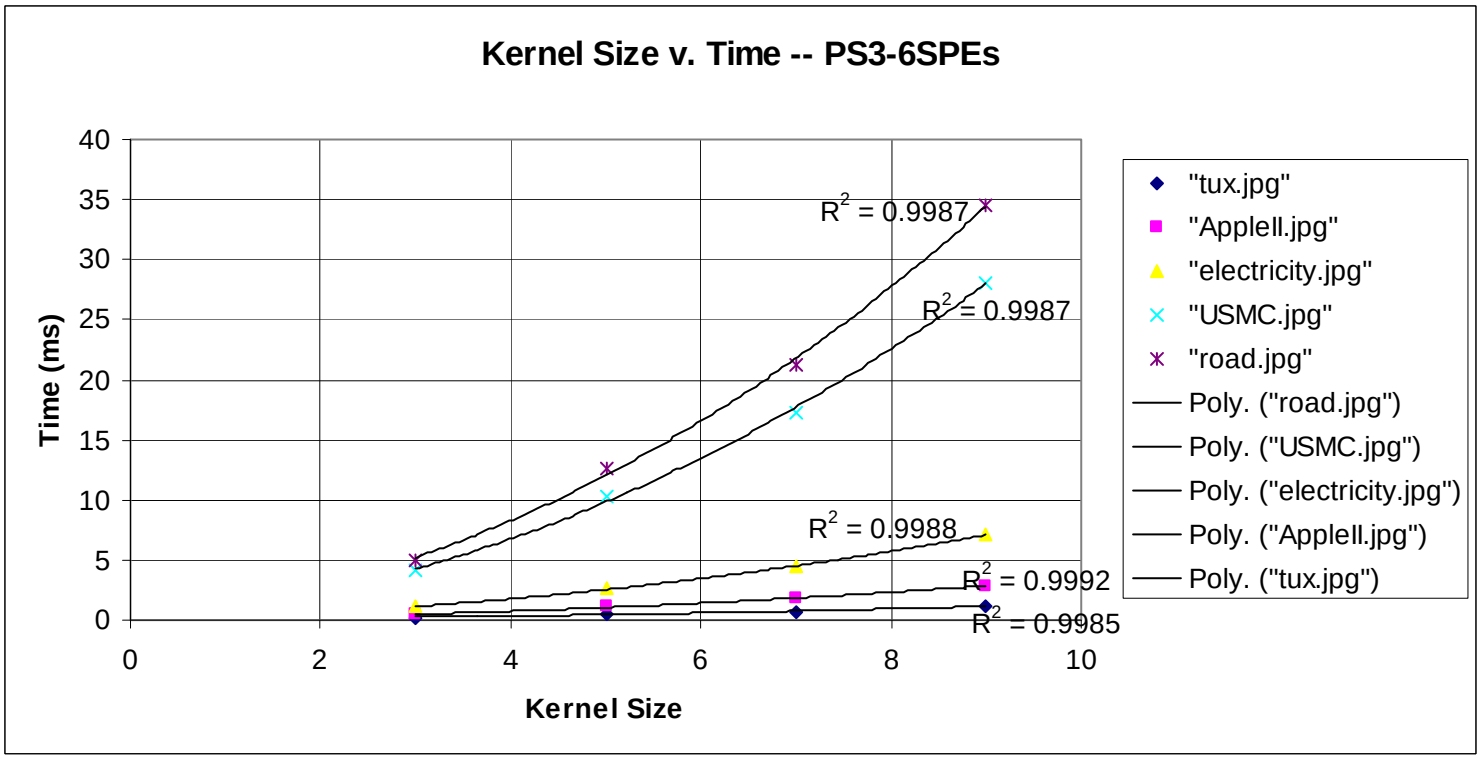


Figure 21: Kernel Size v. Time constant Implementation—PS3-6SPUs

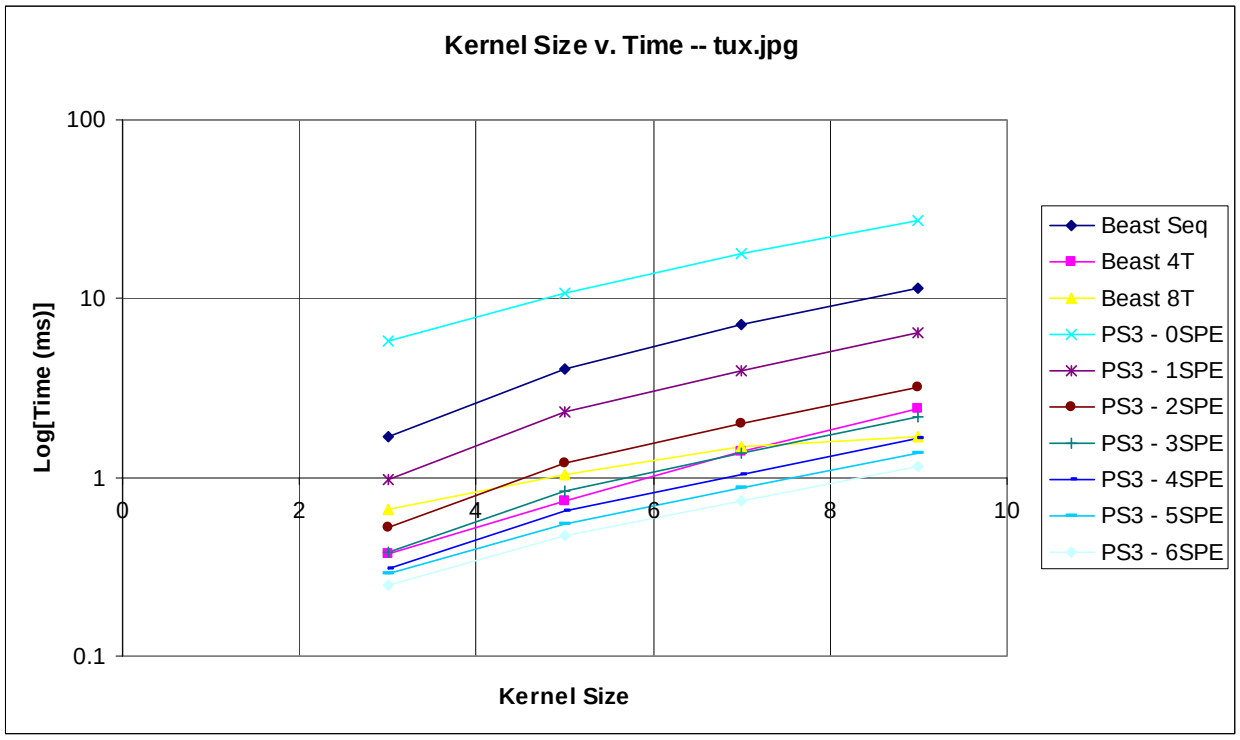


Figure 22: Kernel Size v. Time constant Image—"tux.jpg"

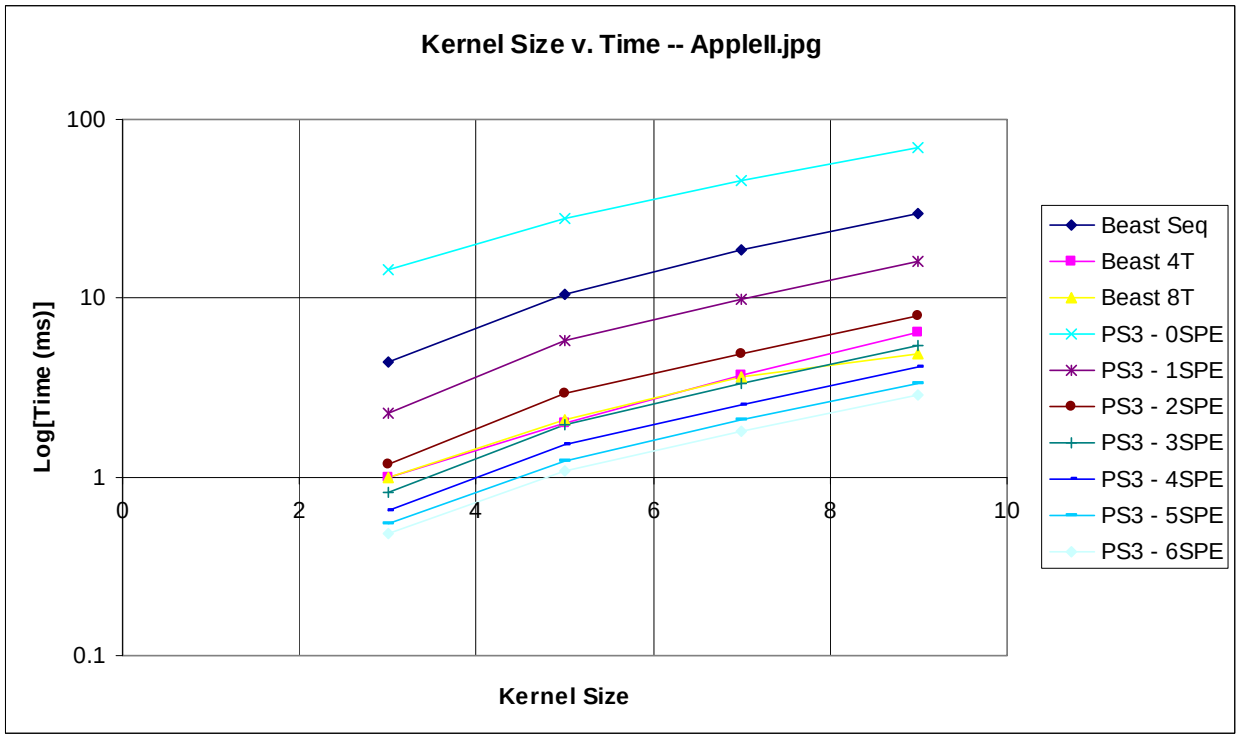


Figure 23: Kernel Size v. Time constant Image—"AppleII.jpg"

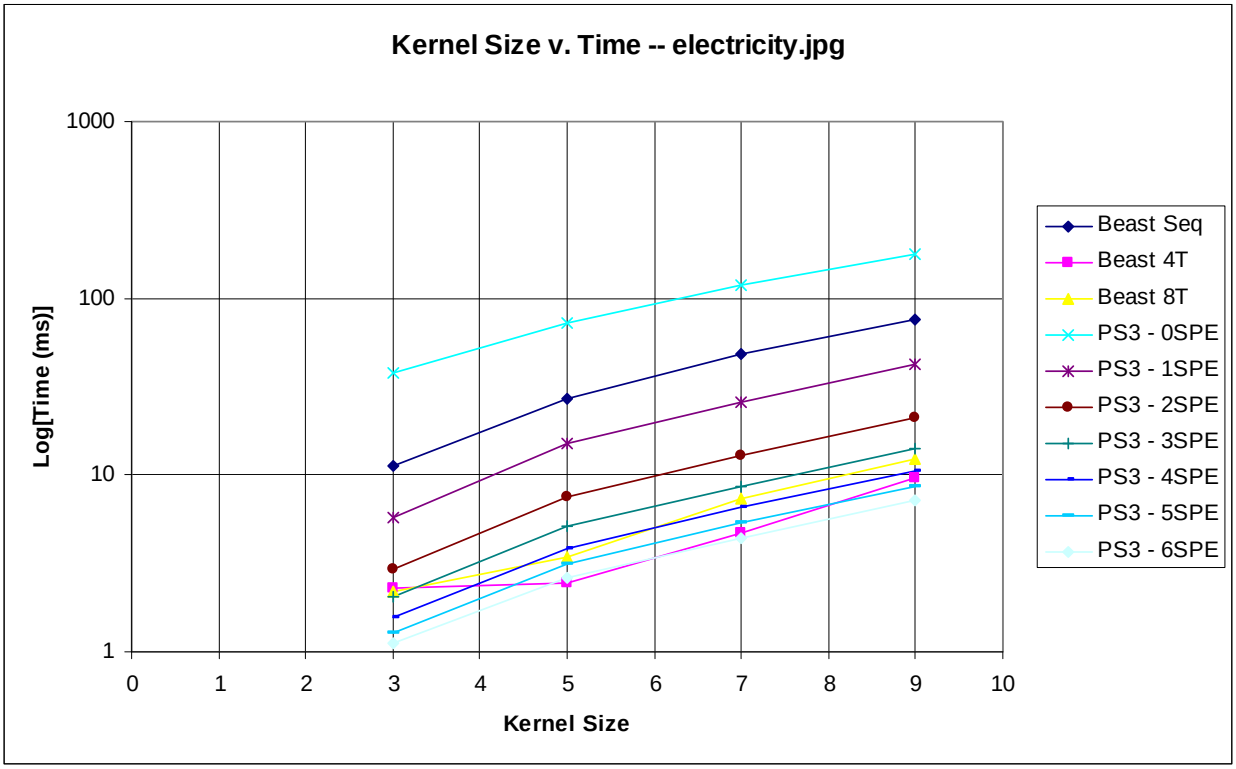


Figure 24: Kernel Size v. Time constant Image—"electricity.jpg"

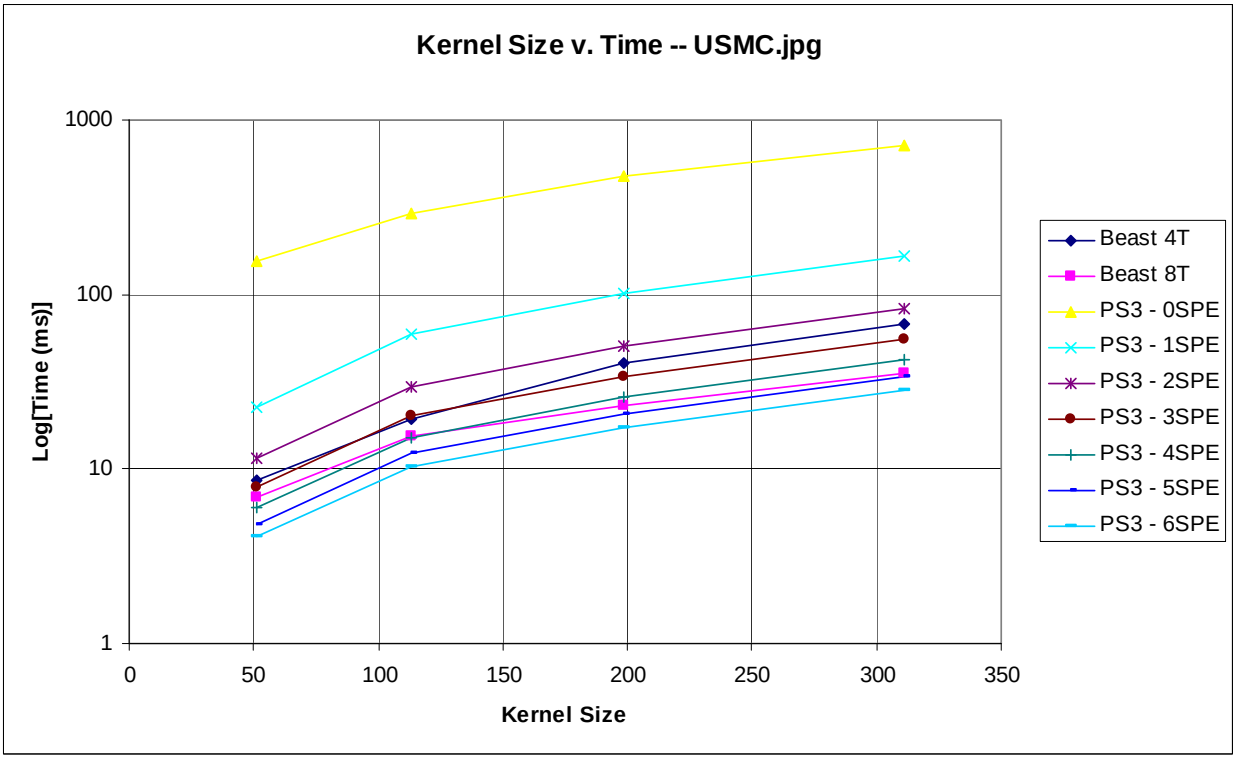


Figure 25: Kernel Size v. Time constant Image—"USMC.jpg"

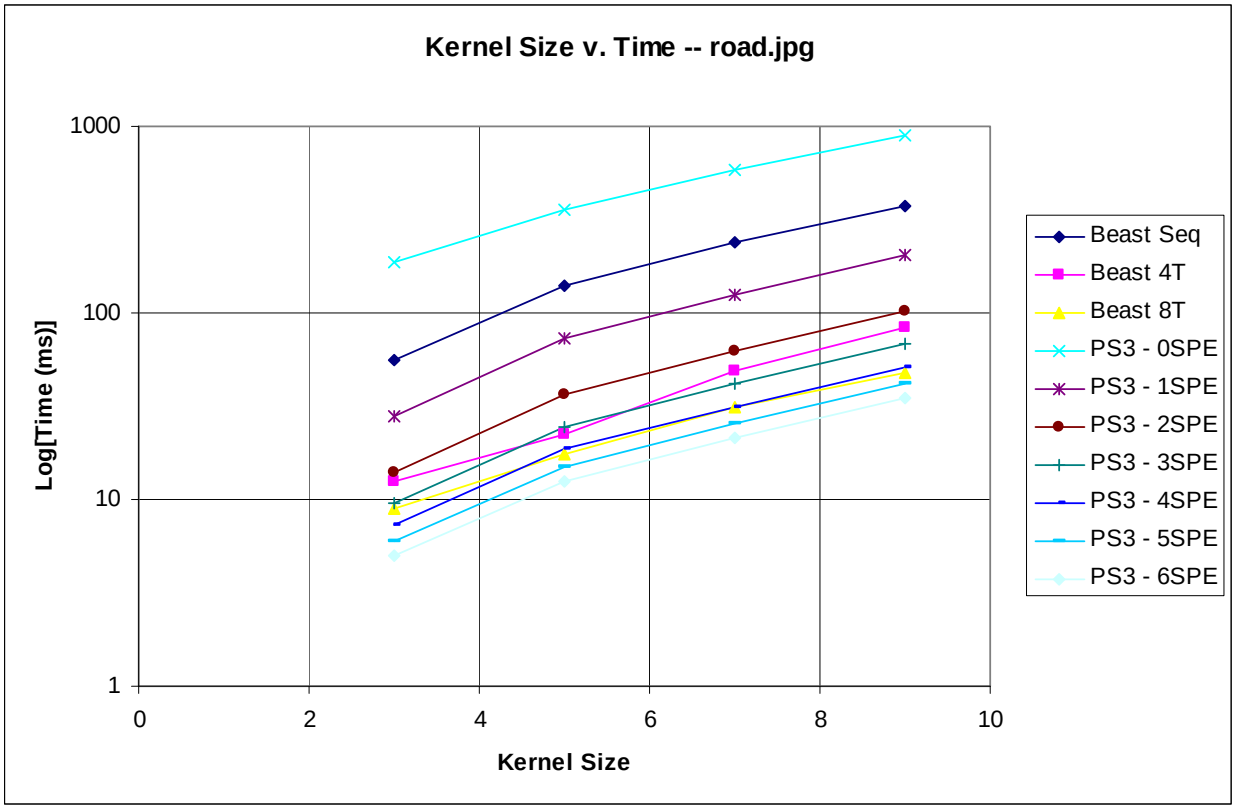


Figure 26: Kernel Size v. Time constant Image—"road.jpg"

The above series of graphs, where Implementation (ie. Sequential, 4 Threaded, 8 Threaded, etc.) is held constant, show that there is indeed a clearly defined quadratic relation between kernel size and execution time. This relationship is entirely expected and holds true for all implementations of the edge detection algorithm. The above series of graphs where Image is held constant for each graph have execution time plotted in a Log scale on the y-axis. This was done to emphasize the relative performance between the execution times of the various implementations shown in series as the kernel size is varied. The general conclusions that can be drawn from this series of graphs are:

The majority of the Cell's computing power comes from the SPE's. When 0 SPEs are utilized, for all kernel sizes and all images, the PlayStation3 actually performs significantly worse than the Sequential algorithm running on the Beast. This is consistent with expectations as the PPU in the Cell is only a modestly powerful processor which is competing with the top of the line x86 processors found in the Beast. However, even with only one SPE engaged, the PlayStation3 immediately outperforms the Sequential version on the Beast. However the 4-threaded and 8-threaded versions on the Beast continue to outperform the Cell processor until 3 and 5 SPEs respectively are engaged. This trend of relative performance holds true for all of the images tested. Additionally, the decreased spacing between the series lines representing the Cell processor as additional SPEs are added, indicates diminishing returns with the introduction of additional processors. This is consistent with Amdahl's law.

4.1.4. Test Results – Image Size v. Execution Time

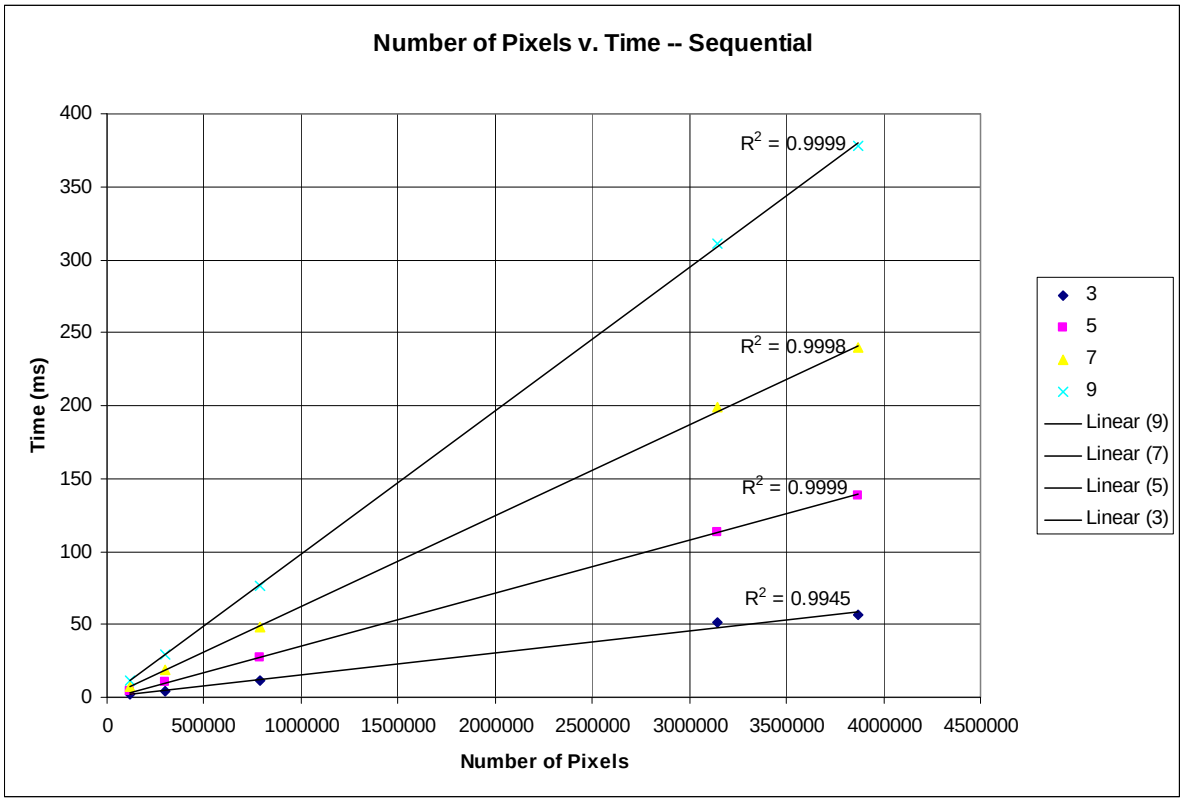


Figure 27: Image Size v. Time constant Implementation—Sequential

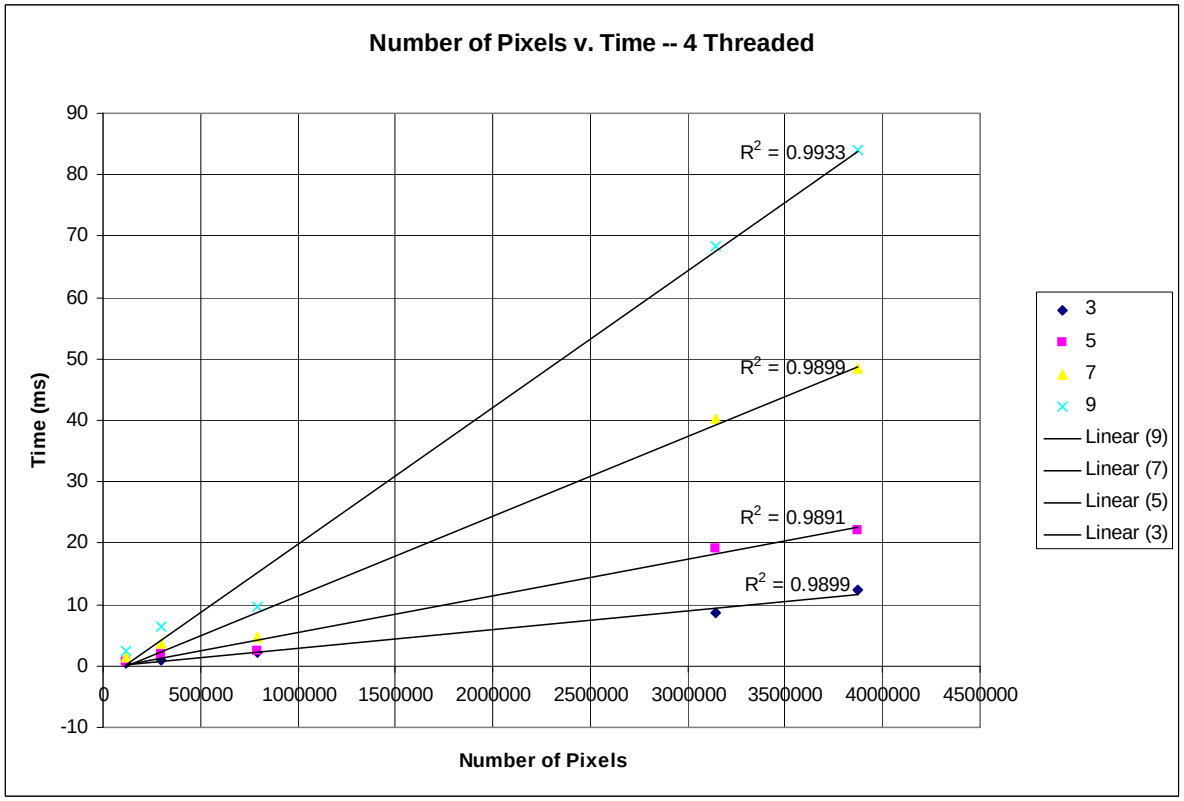


Figure 28: Image Size v. Time constant Implementation —4 Threaded

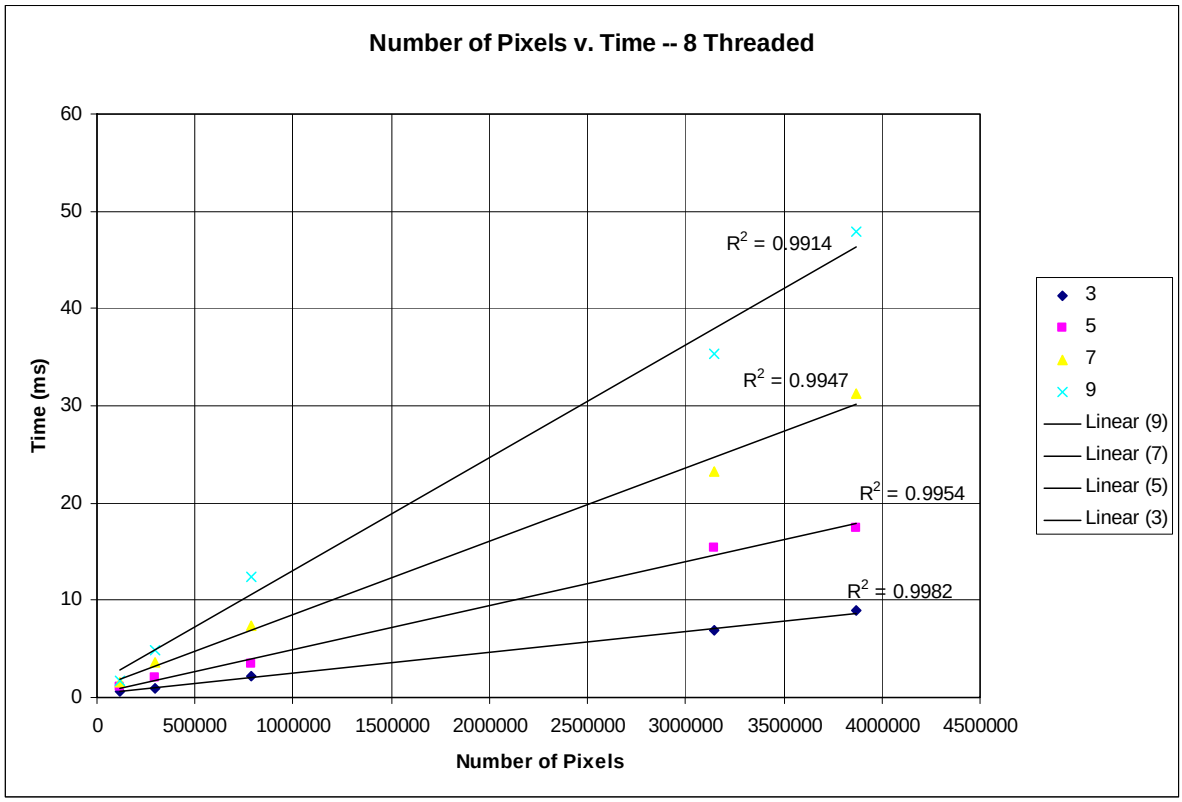


Figure 29: Image Size v. Time constant Implementation —8 Threaded

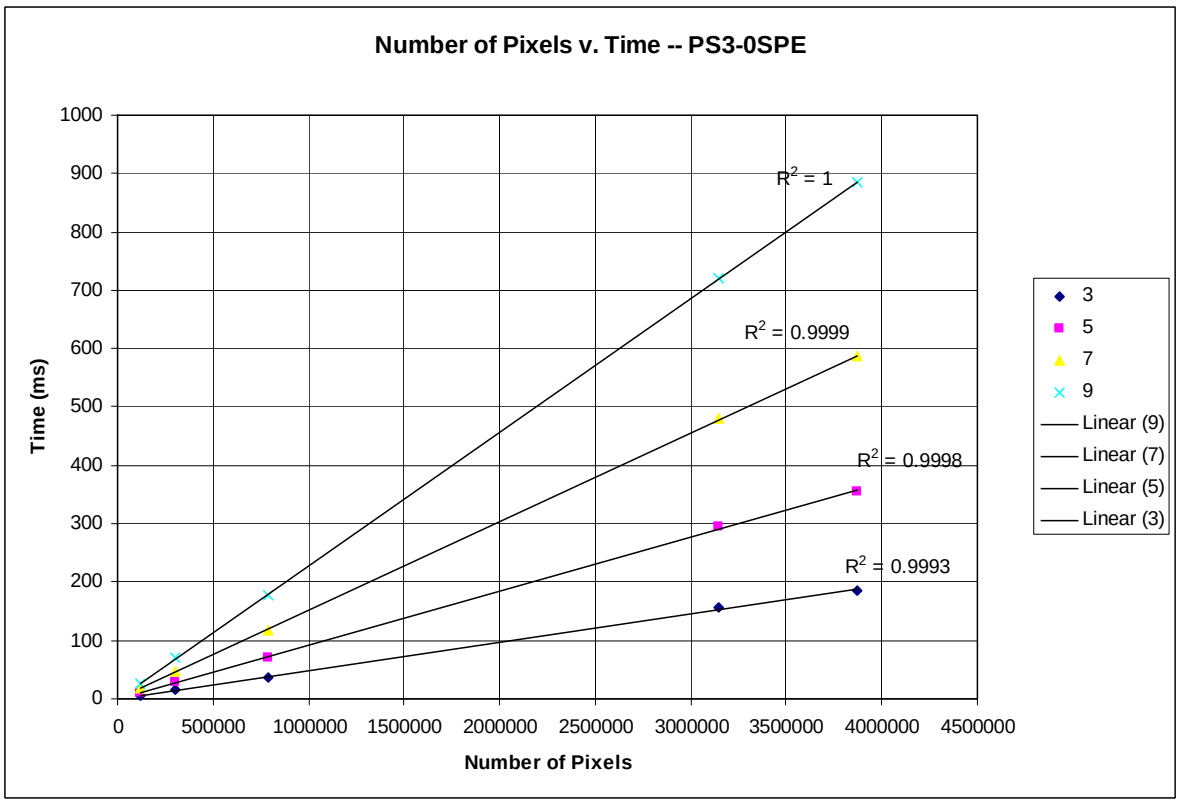


Figure 30: Image Size v. Time constant Implementation —PS3 0-SPEs

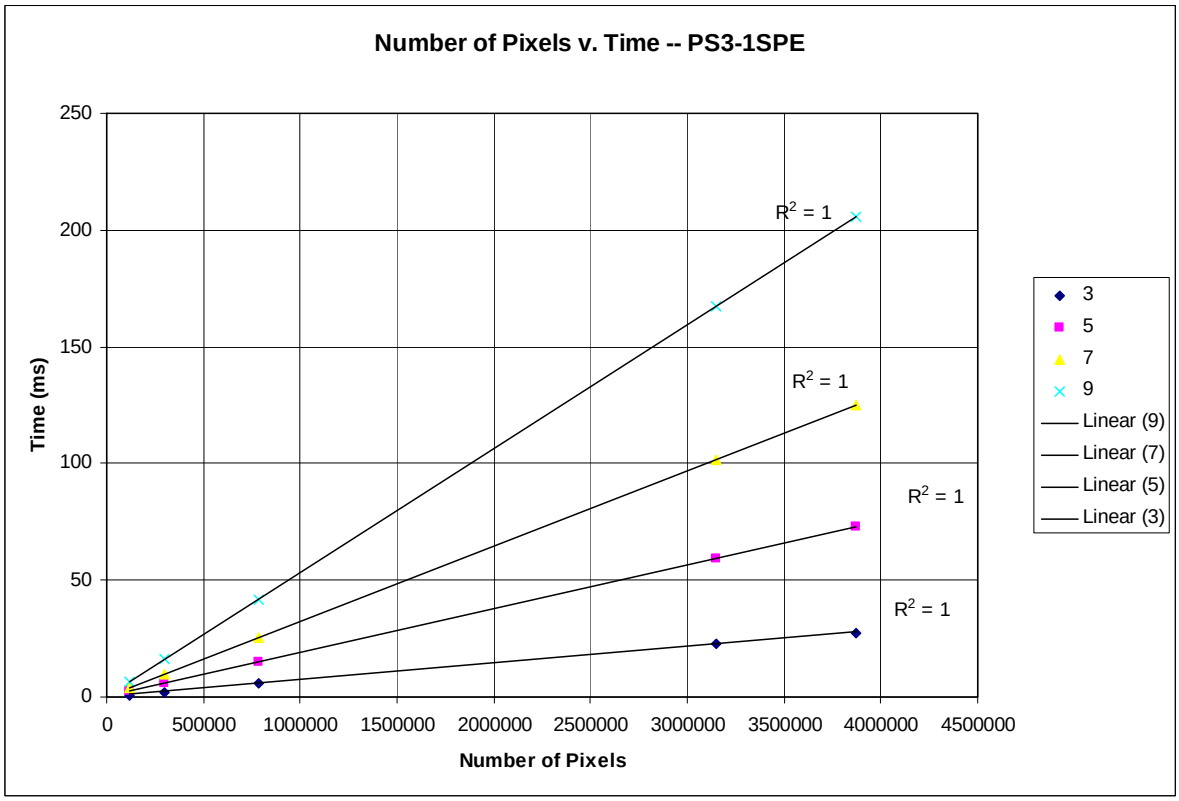


Figure 31: Image Size v. Time constant Implementation —PS3 1-SPEs

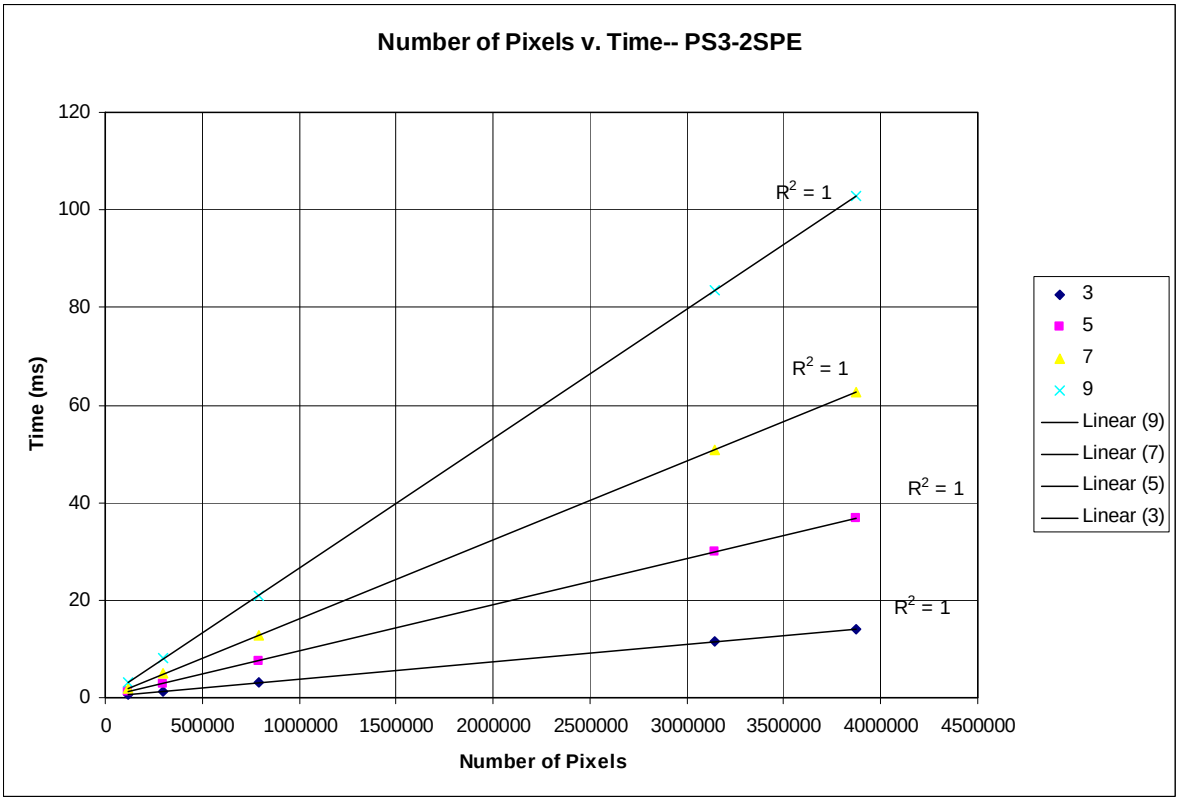


Figure 32: Image Size v. Time constant Implementation —PS3 2-SPEs

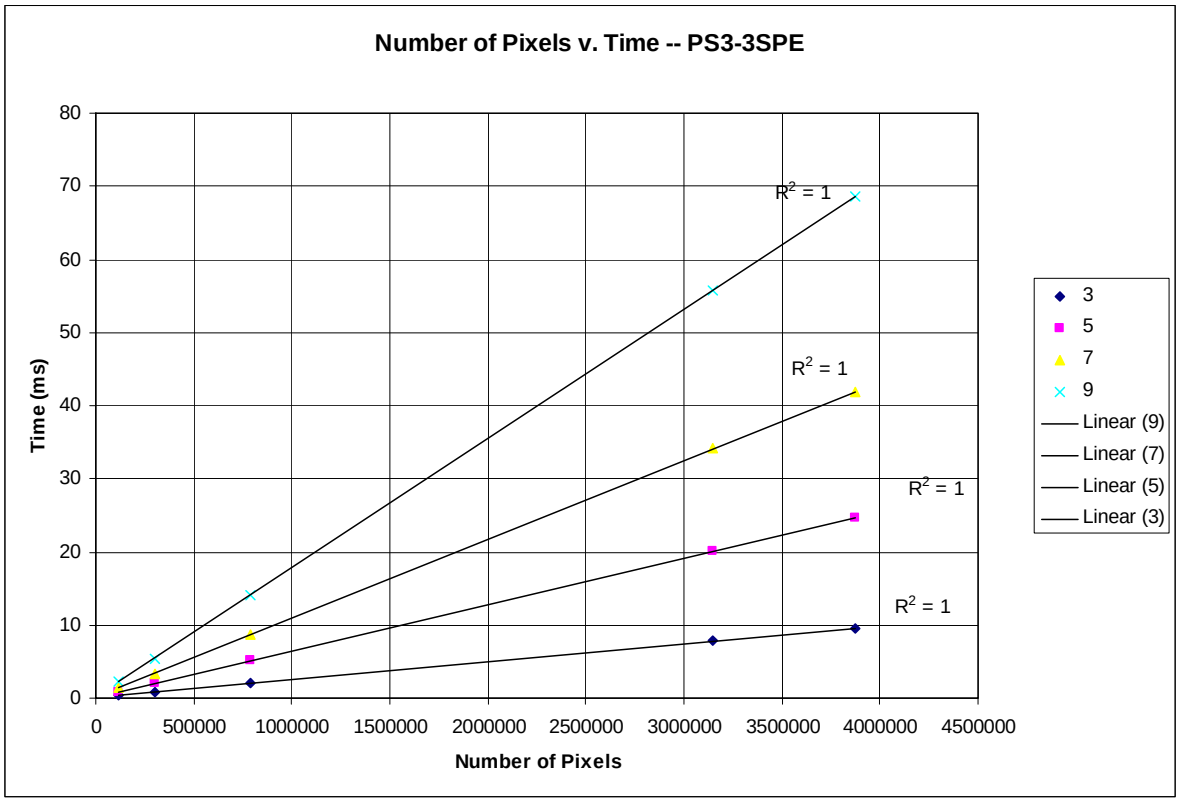


Figure 33: Image Size v. Time constant Implementation —PS3 3-SPEs

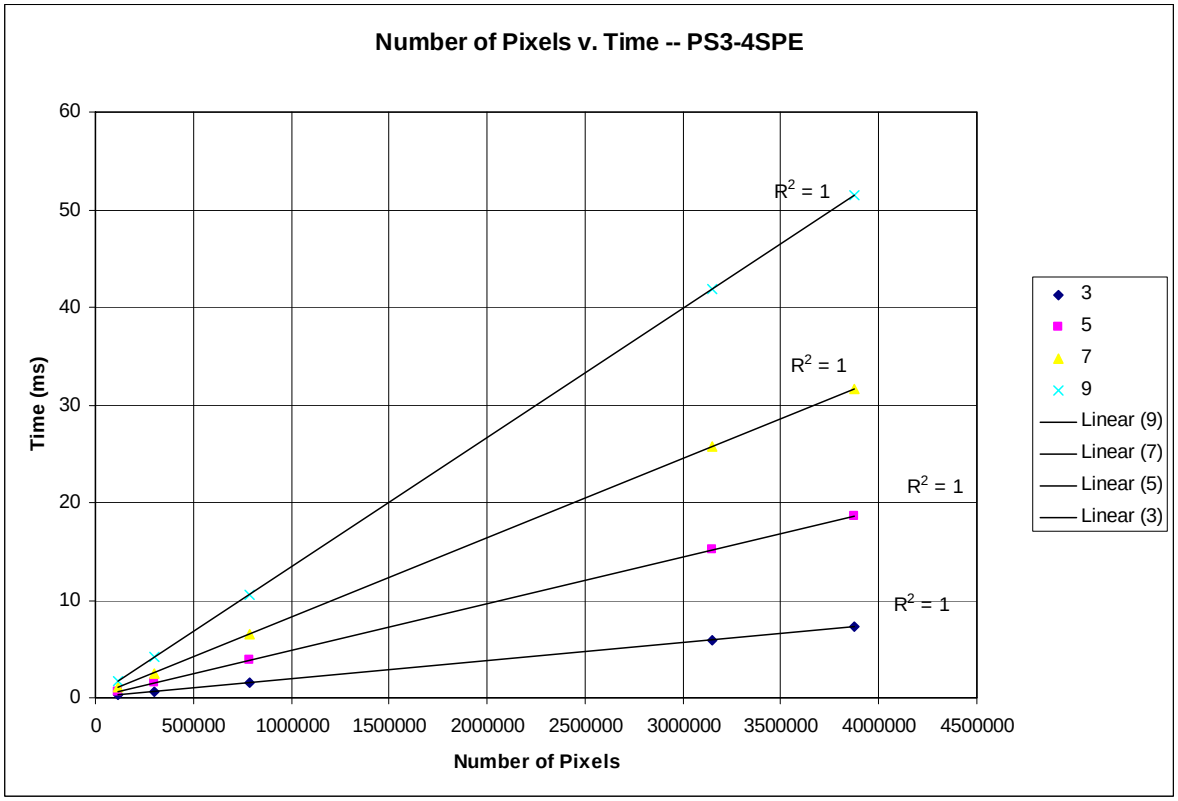


Figure 34: Image Size v. Time constant Implementation —PS3 4-SPEs

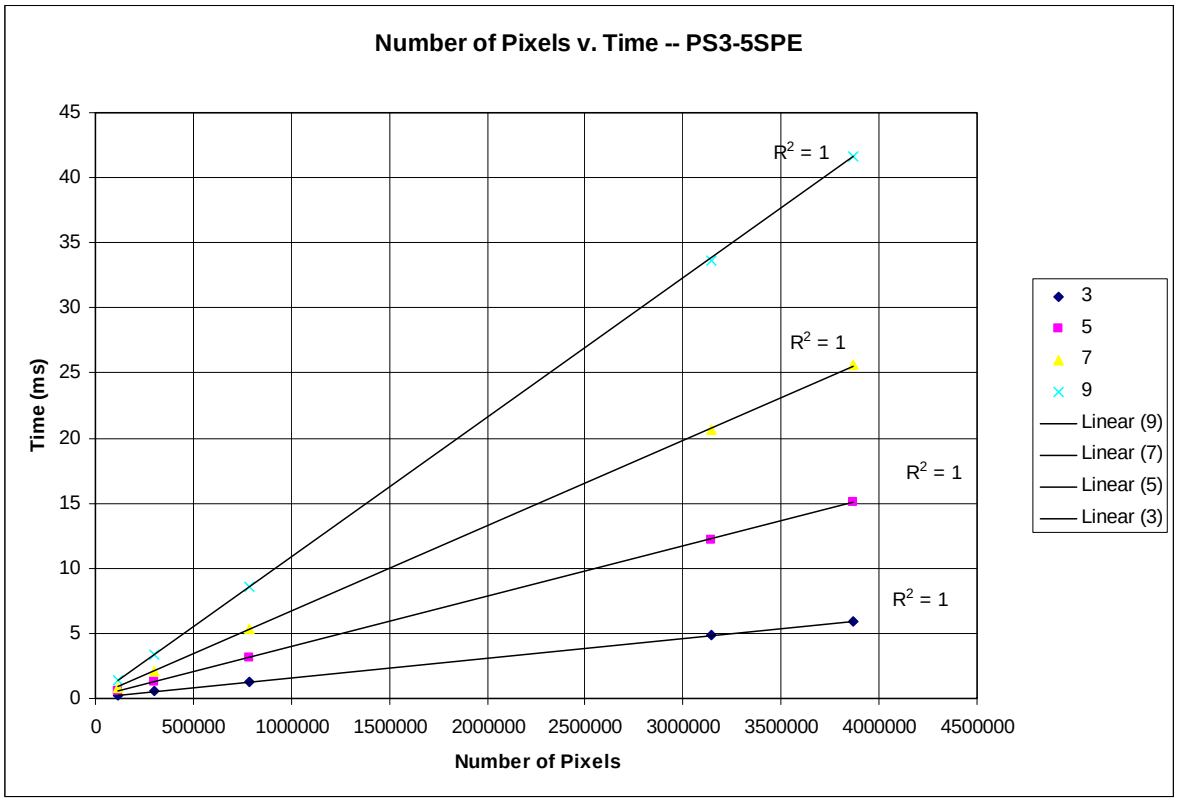


Figure 35: Image Size v. Time constant Implementation —PS3 5-SPEs

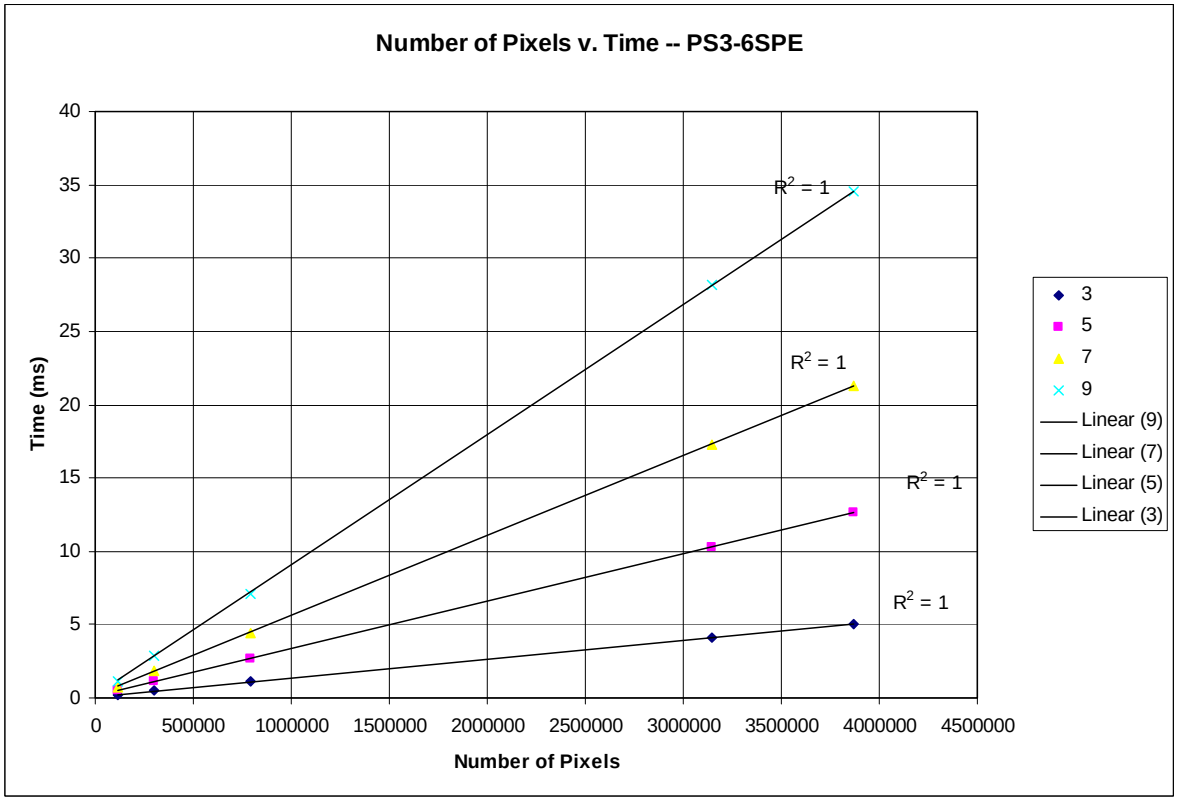


Figure 36: Image Size v. Time constant Implementation —PS3 6-SPEs

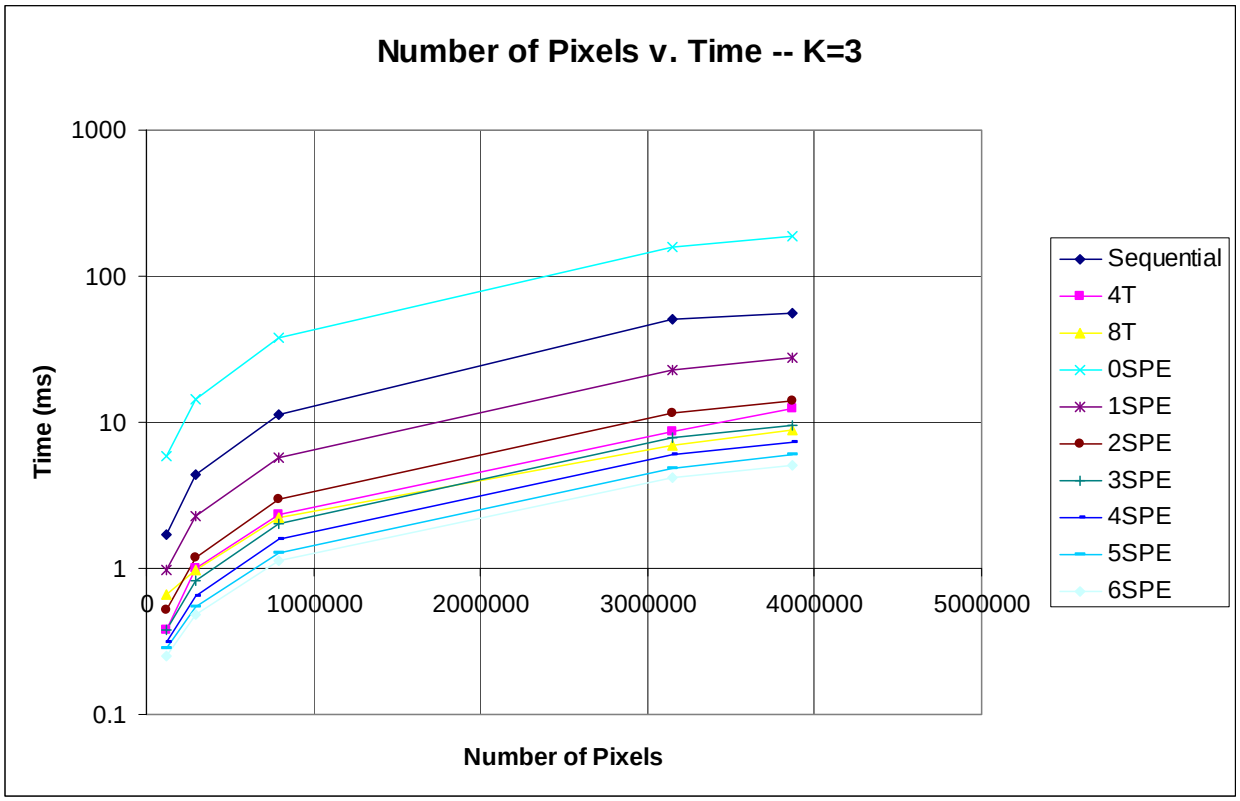


Figure 37: Image Size v. Time constant Kernel Size — K=3

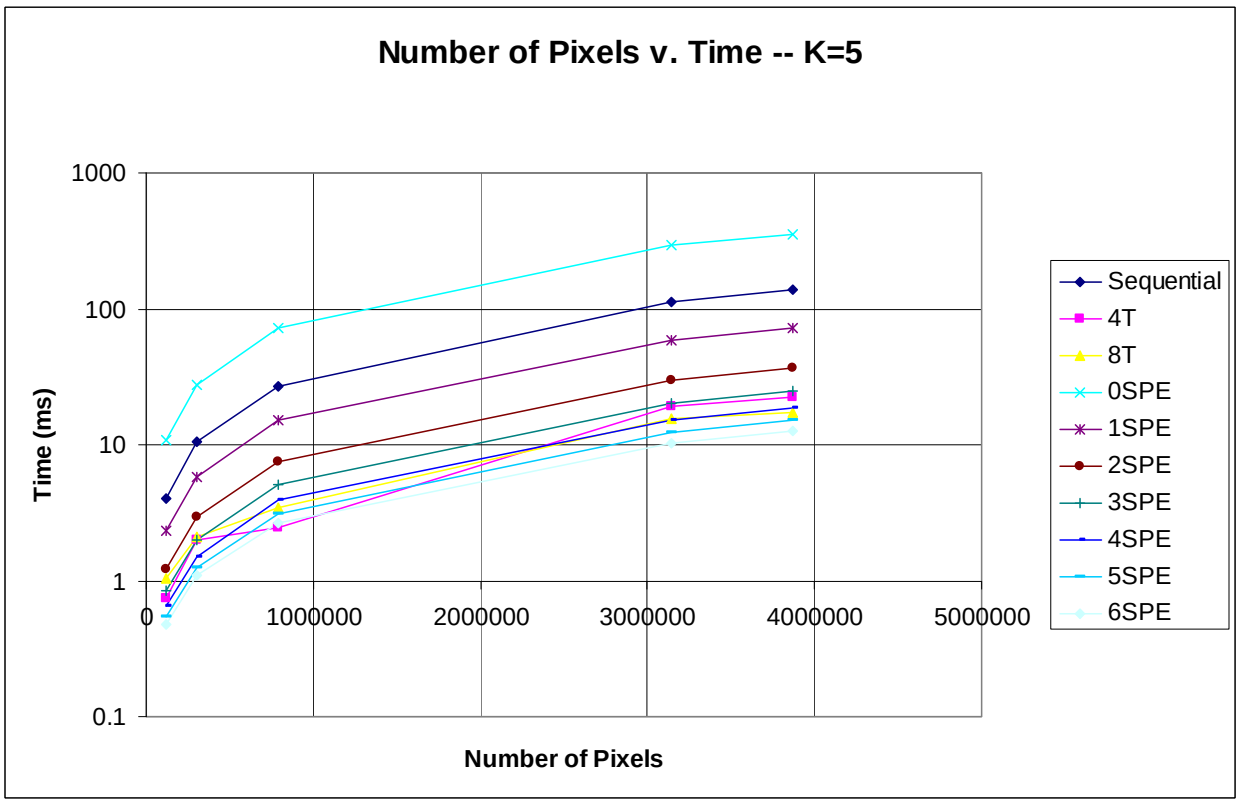


Figure 38: Image Size v. Time constant Kernel Size — K=5

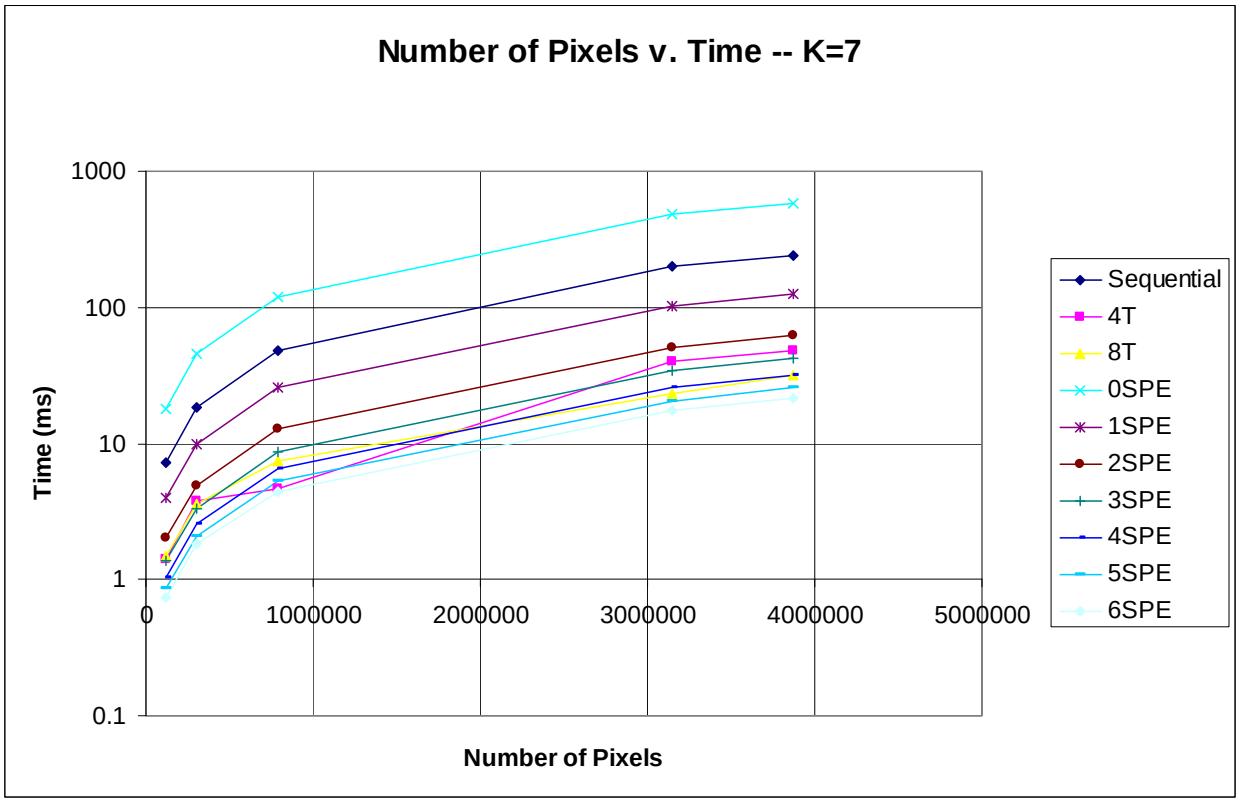


Figure 39: Image Size v. Time constant Kernel Size — K=7

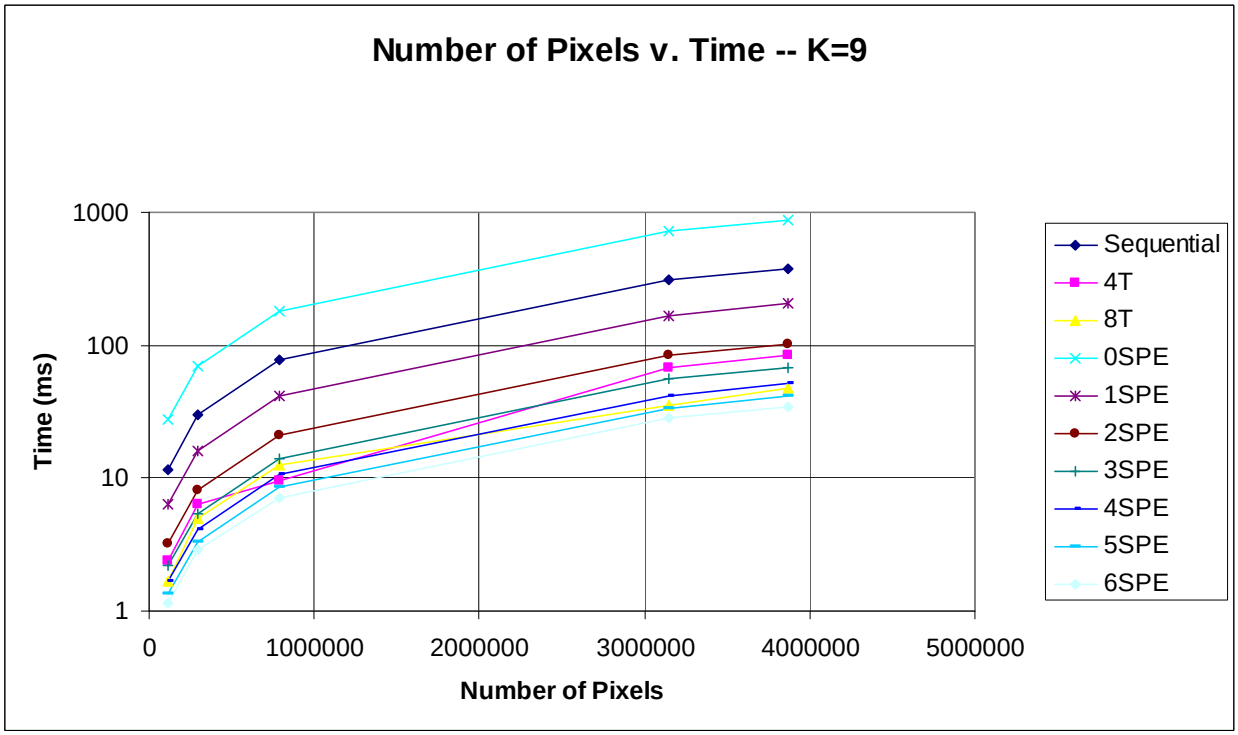
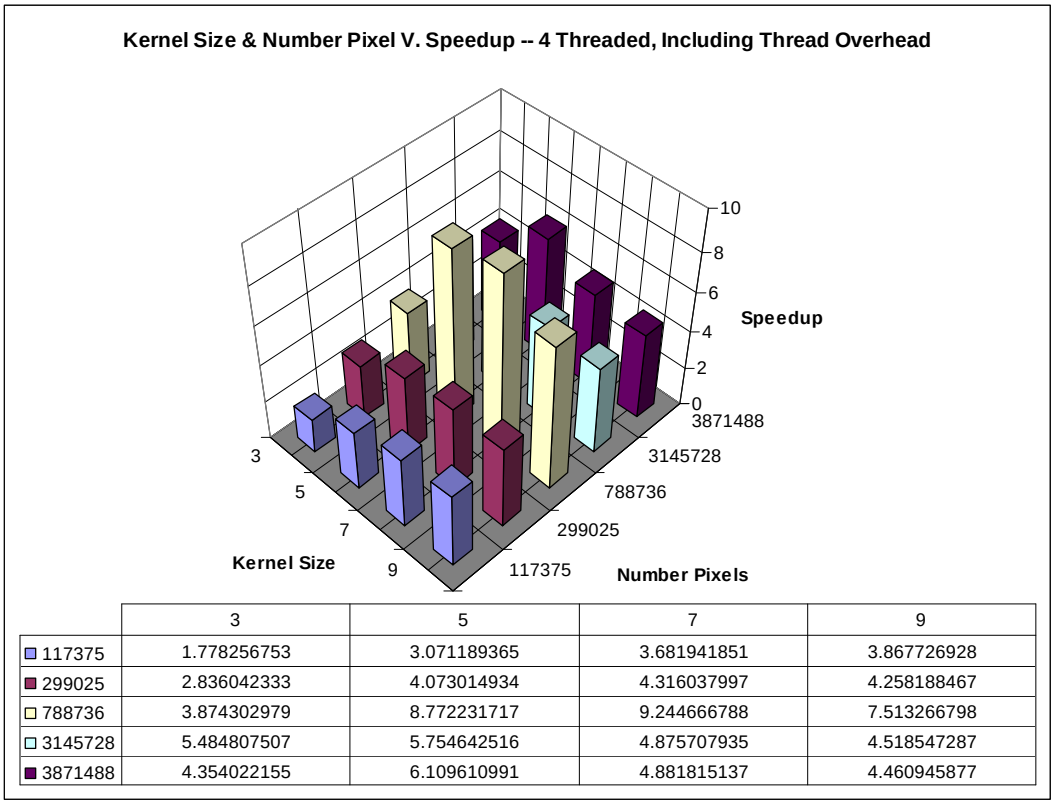


Figure 40: Image Size v. Time constant Kernel Size — K=9

The above series of graphs, where Implementation (ie. Sequential, 4 Threaded, 8 Threaded, etc.) is held constant, show that there is indeed a clearly defined linear relation between image size and execution time. This relationship is entirely expected and holds true for all implementations of the edge detection algorithm. The above series of graphs where Kernel size is held constant for each graph have execution time plotted in a Log scale on the y-axis. This was done to emphasize the relative performance between the execution times of the various implementations shown in series as the kernel size is varied. The general conclusions that can be drawn from this series of graphs are equivalent to the conclusions drawn from the previous Log scale plots:

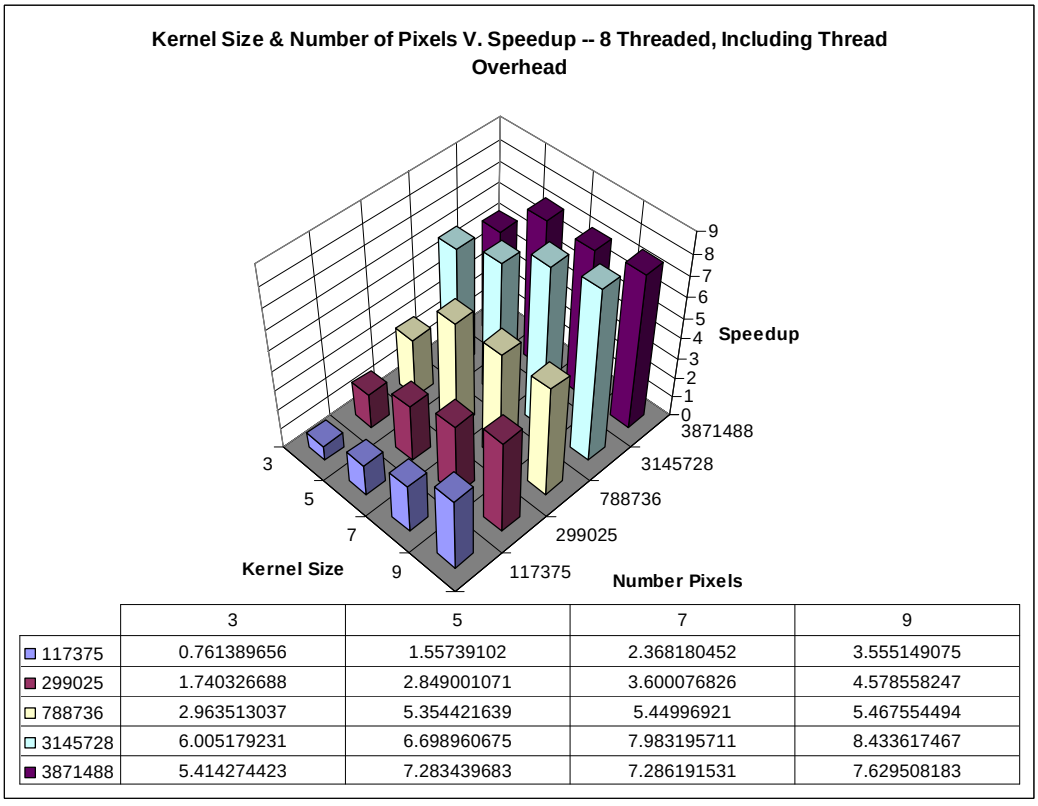
The majority of the Cell's computing power comes from the SPE's. When 0 SPEs are utilized, for all kernel sizes and all images, the PlayStation3 actually performs significantly worse than the Sequential algorithm running on the Beast. This is consistent with expectations as the PPU in the Cell is only a modestly powerful processor which is competing with the top of the line x86 processors found in the Beast. However, even with only one SPE engaged, the PlayStation3 immediately outperforms the Sequential version on the Beast. However the 4-threaded and 8-threaded versions on the Beast continue to outperform the Cell processor until 3 and 5 SPEs respectively are engaged. This trend of relative performance holds true for all of the images tested. Additionally, the decreased spacing between the series lines representing the Cell processor as additional SPEs are added, indicates diminishing returns with the introduction of additional processors. Once again, this is consistent with Amdahl's law.

4.1.5. Test Results – Kernel Size & Image Size v. Speedup



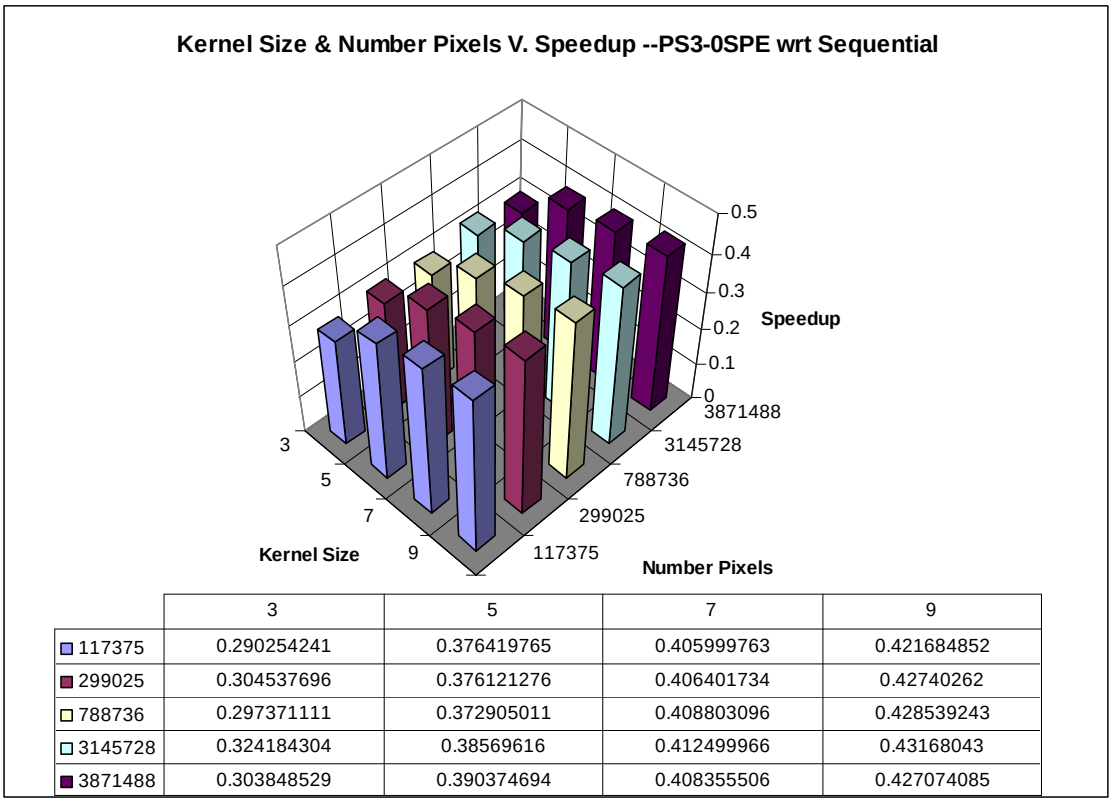
	3	5	7	9
117375	1.778256753	3.071189365	3.681941851	3.867726928
299025	2.836042333	4.073014934	4.316037997	4.258188467
788736	3.874302979	8.772231717	9.244666788	7.513266798
3145728	5.484807507	5.754642516	4.875707935	4.518547287
3871488	4.354022155	6.109610991	4.881815137	4.460945877

Figure 41: Kernel Size & Image Size v. Speedup – 4 Threads



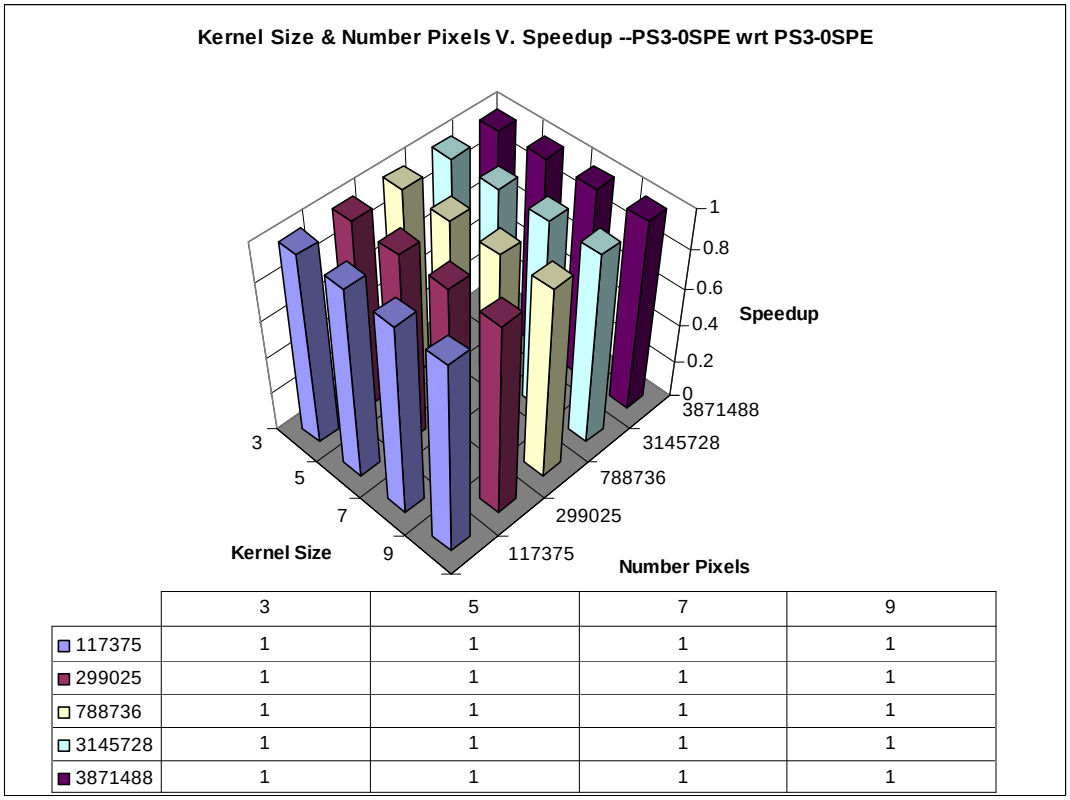
	3	5	7	9
117375	0.761389656	1.55739102	2.368180452	3.555149075
299025	1.740326688	2.849001071	3.600076826	4.578558247
788736	2.963513037	5.354421639	5.44996921	5.467554494
3145728	6.005179231	6.698960675	7.983195711	8.433617467
3871488	5.414274423	7.283439683	7.286191531	7.629508183

Figure 42: Kernel Size & Image Size v. Speedup – 8 Threads



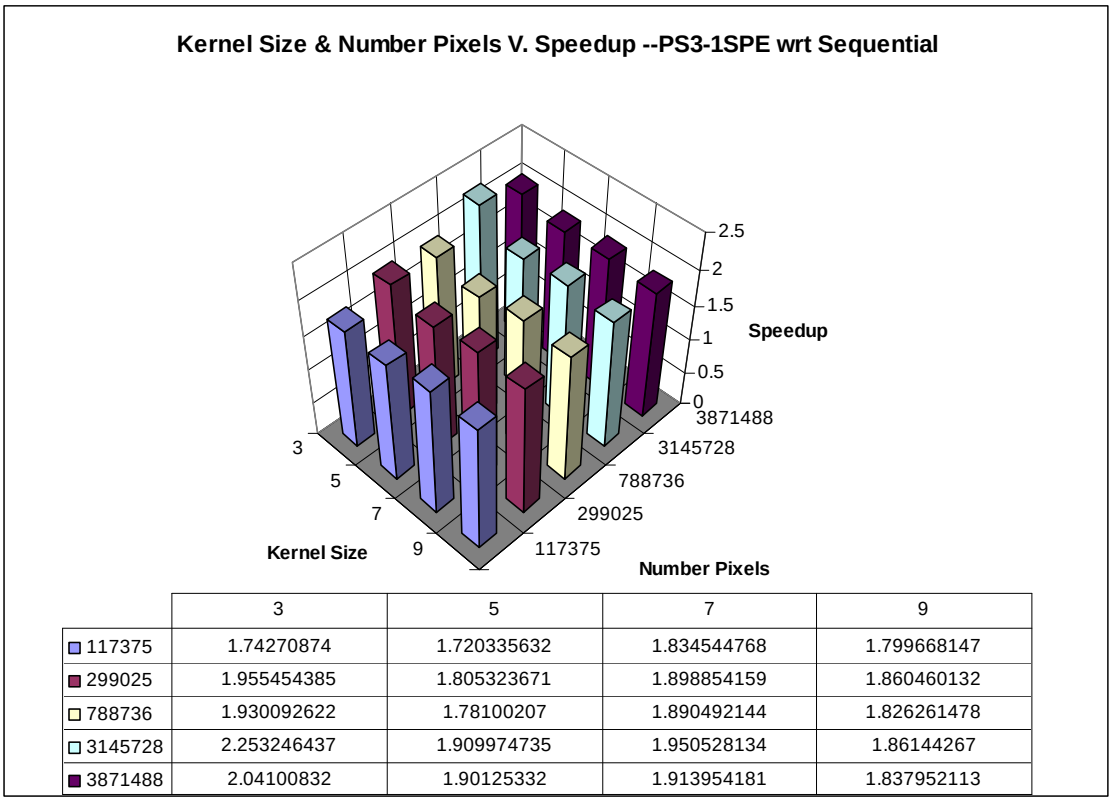
	3	5	7	9
117375	0.290254241	0.376419765	0.405999763	0.421684852
299025	0.304537696	0.376121276	0.406401734	0.42740262
788736	0.297371111	0.372905011	0.408803096	0.428539243
3145728	0.324184304	0.38569616	0.412499966	0.43168043
3871488	0.303848529	0.390374694	0.408355506	0.427074085

Figure 43: Kernel Size & Image Size v. Speedup – PS3-0SPE WRT Sequential



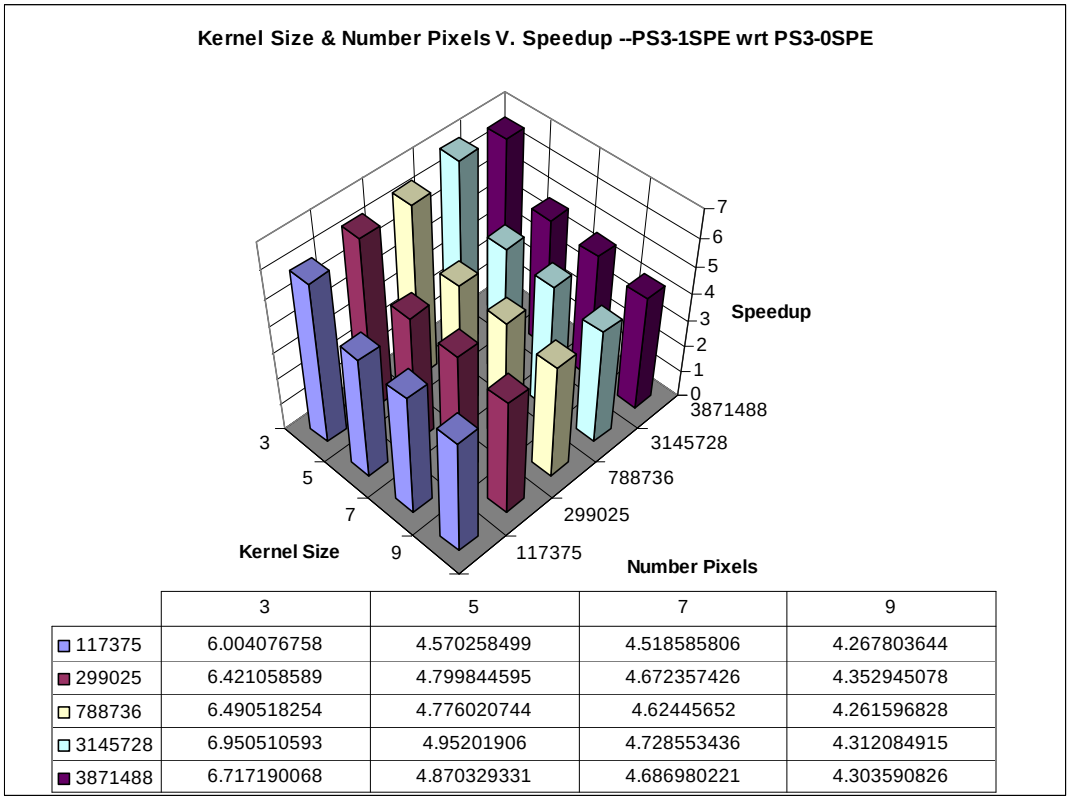
	3	5	7	9
117375	1	1	1	1
299025	1	1	1	1
788736	1	1	1	1
3145728	1	1	1	1
3871488	1	1	1	1

Figure 44: Kernel Size & Image Size v. Speedup – PS3-0SPE WRT PS3-0SPE



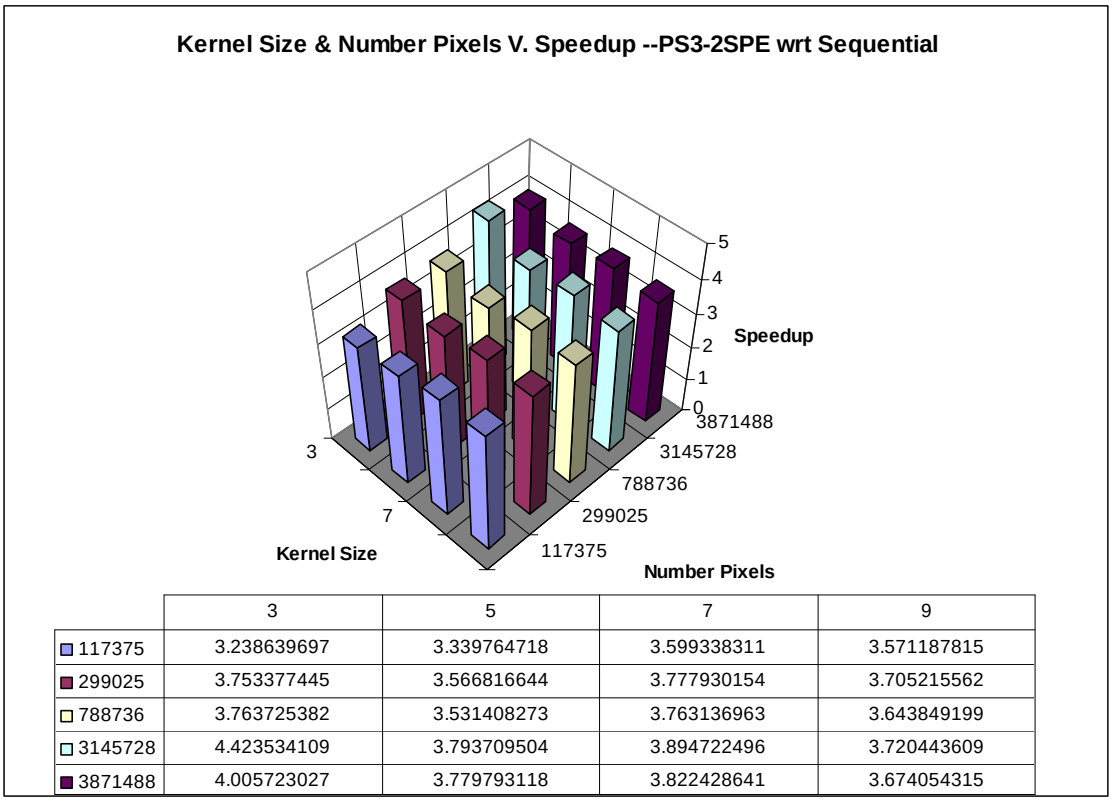
	3	5	7	9
117375	1.74270874	1.720335632	1.834544768	1.799668147
299025	1.955454385	1.805323671	1.898854159	1.860460132
788736	1.930092622	1.78100207	1.890492144	1.826261478
3145728	2.253246437	1.909974735	1.950528134	1.86144267
3871488	2.04100832	1.90125332	1.913954181	1.837952113

Figure 45: Kernel Size & Image Size v. Speedup – PS3-1SPE WRT Sequential



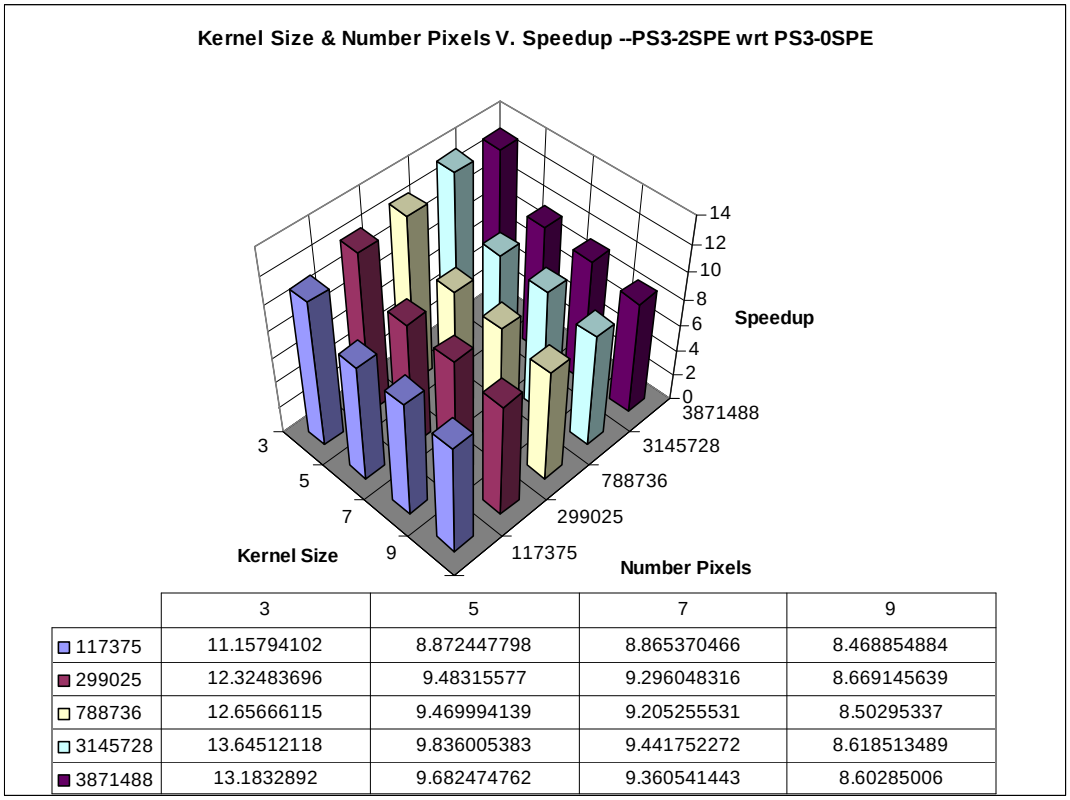
	3	5	7	9
117375	6.004076758	4.570258499	4.518585806	4.267803644
299025	6.421058589	4.799844595	4.672357426	4.352945078
788736	6.490518254	4.776020744	4.62445652	4.261596828
3145728	6.950510593	4.95201906	4.728553436	4.312084915
3871488	6.717190068	4.870329331	4.686980221	4.303590826

Figure 46: Kernel Size & Image Size v. Speedup – PS3-1SPE WRT PS3-0SPE



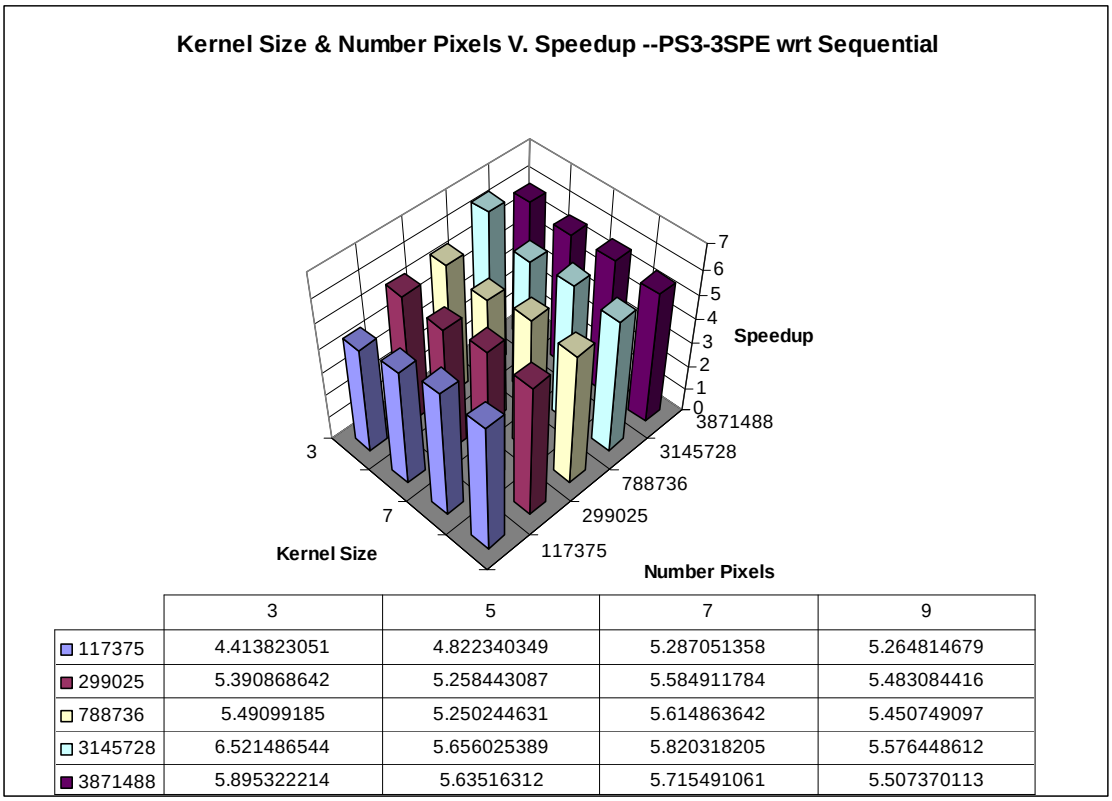
	3	5	7	9
117375	3.238639697	3.339764718	3.599338311	3.571187815
299025	3.753377445	3.566816644	3.777930154	3.705215562
788736	3.763725382	3.531408273	3.763136963	3.643849199
3145728	4.423534109	3.793709504	3.894722496	3.720443609
3871488	4.005723027	3.779793118	3.822428641	3.674054315

Figure 47: Kernel Size & Image Size v. Speedup – PS3-2SPE WRT Sequential



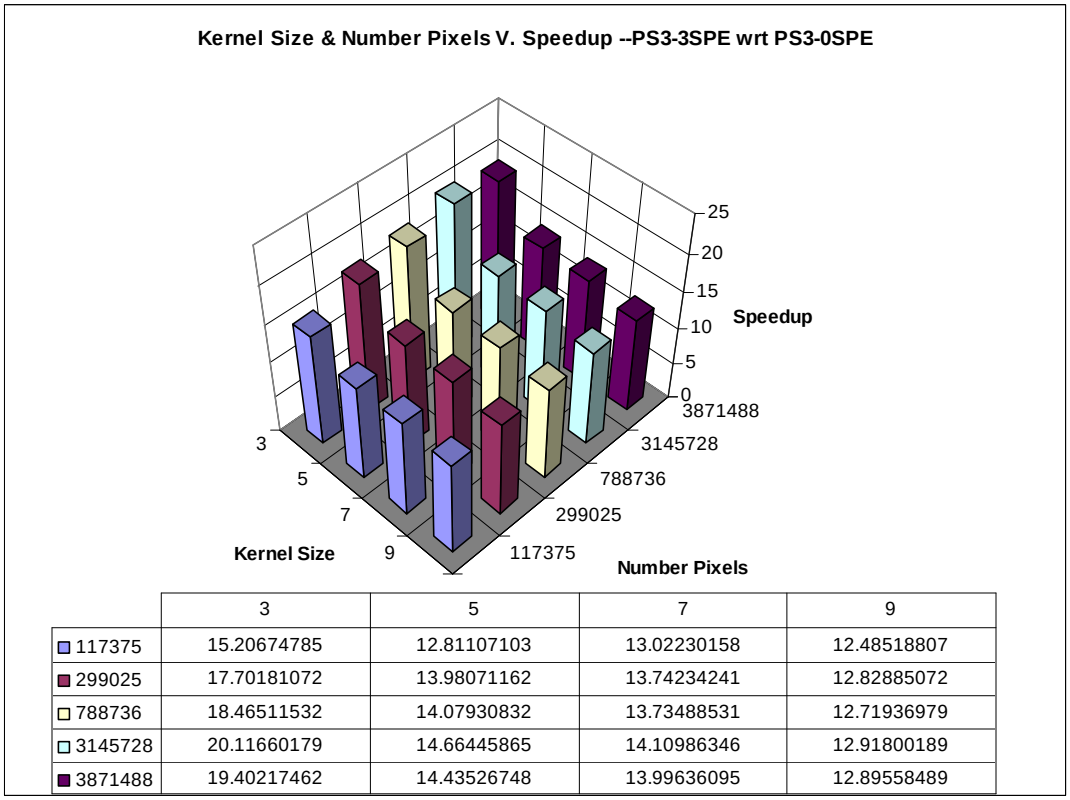
	3	5	7	9
117375	11.15794102	8.872447798	8.865370466	8.468854884
299025	12.32483696	9.48315577	9.296048316	8.669145639
788736	12.65666115	9.469994139	9.205255531	8.50295337
3145728	13.64512118	9.836005383	9.441752272	8.618513489
3871488	13.1832892	9.682474762	9.360541443	8.60285006

Figure 48: Kernel Size & Image Size v. Speedup – PS3-2SPE WRT PS3-0SPE



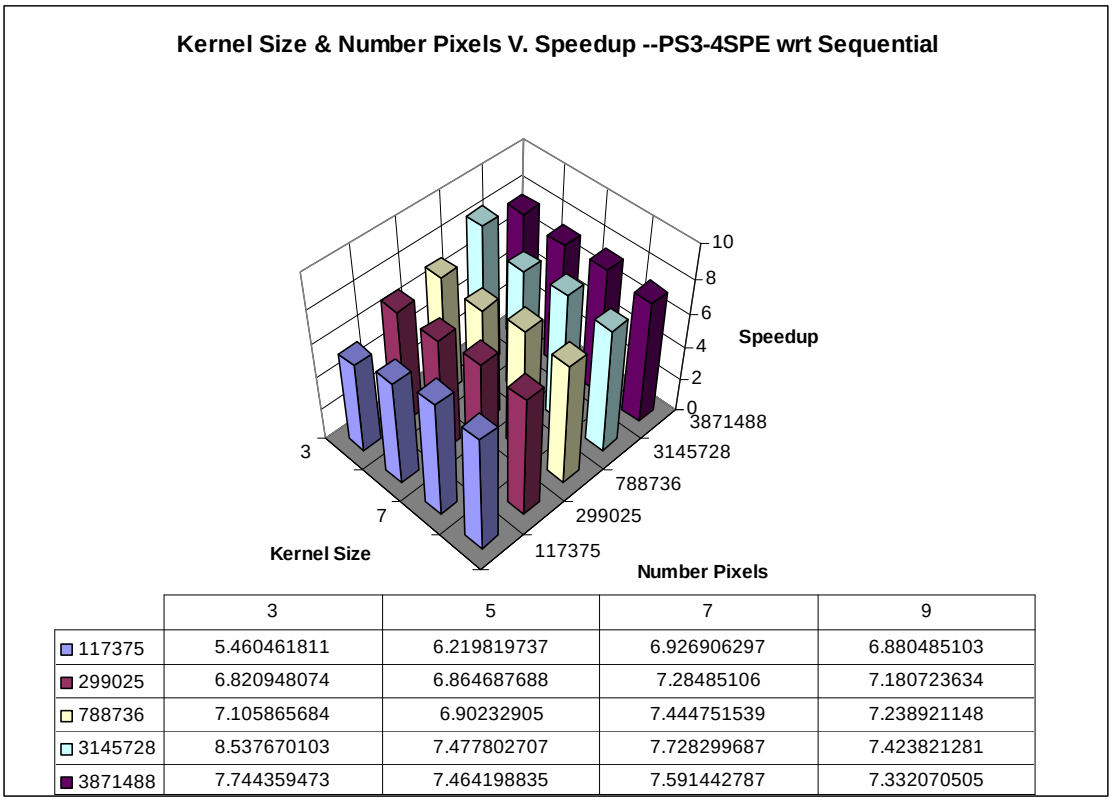
	3	5	7	9
117375	4.413823051	4.822340349	5.287051358	5.264814679
299025	5.390868642	5.258443087	5.584911784	5.483084416
788736	5.49099185	5.250244631	5.614863642	5.450749097
3145728	6.521486544	5.656025389	5.820318205	5.576448612
3871488	5.895322214	5.63516312	5.715491061	5.507370113

Figure 49: Kernel Size & Image Size v. Speedup – PS3-3SPE WRT Sequential



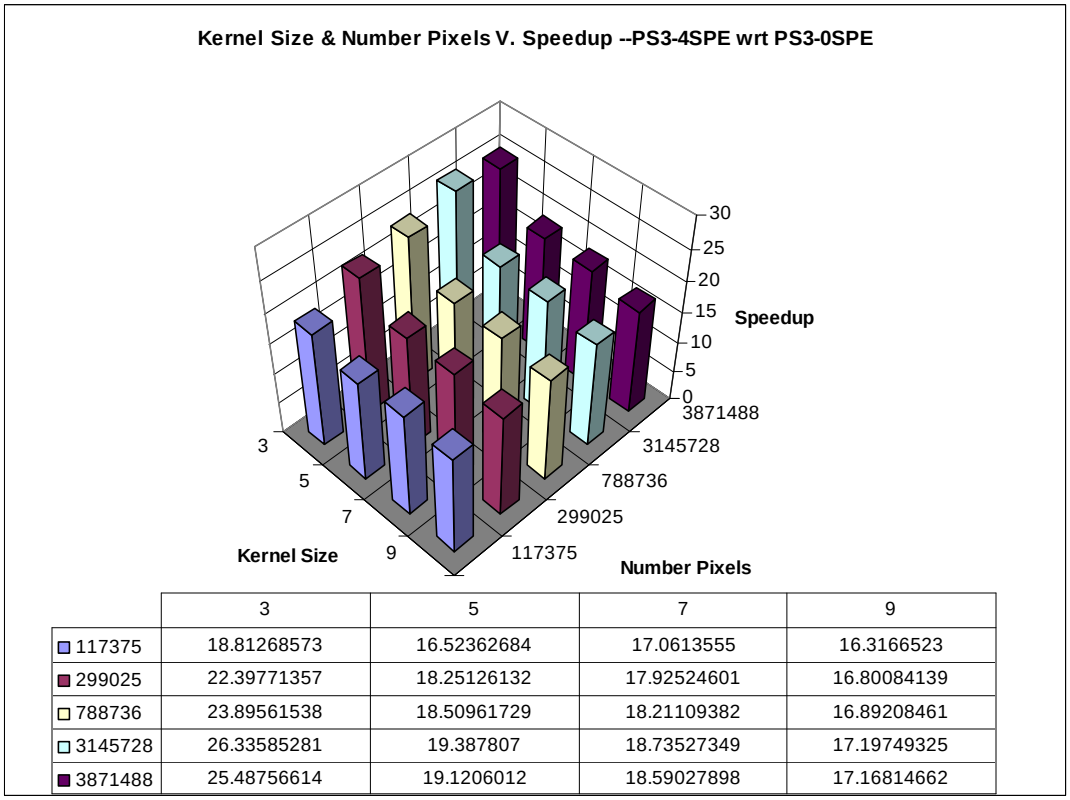
	3	5	7	9
117375	15.20674785	12.81107103	13.02230158	12.48518807
299025	17.70181072	13.98071162	13.74234241	12.82885072
788736	18.46511532	14.07930832	13.73488531	12.71936979
3145728	20.11660179	14.66445865	14.10986346	12.91800189
3871488	19.40217462	14.43526748	13.99636095	12.89558489

Figure 50: Kernel Size & Image Size v. Speedup – PS3-3SPE WRT PS3-0SPE



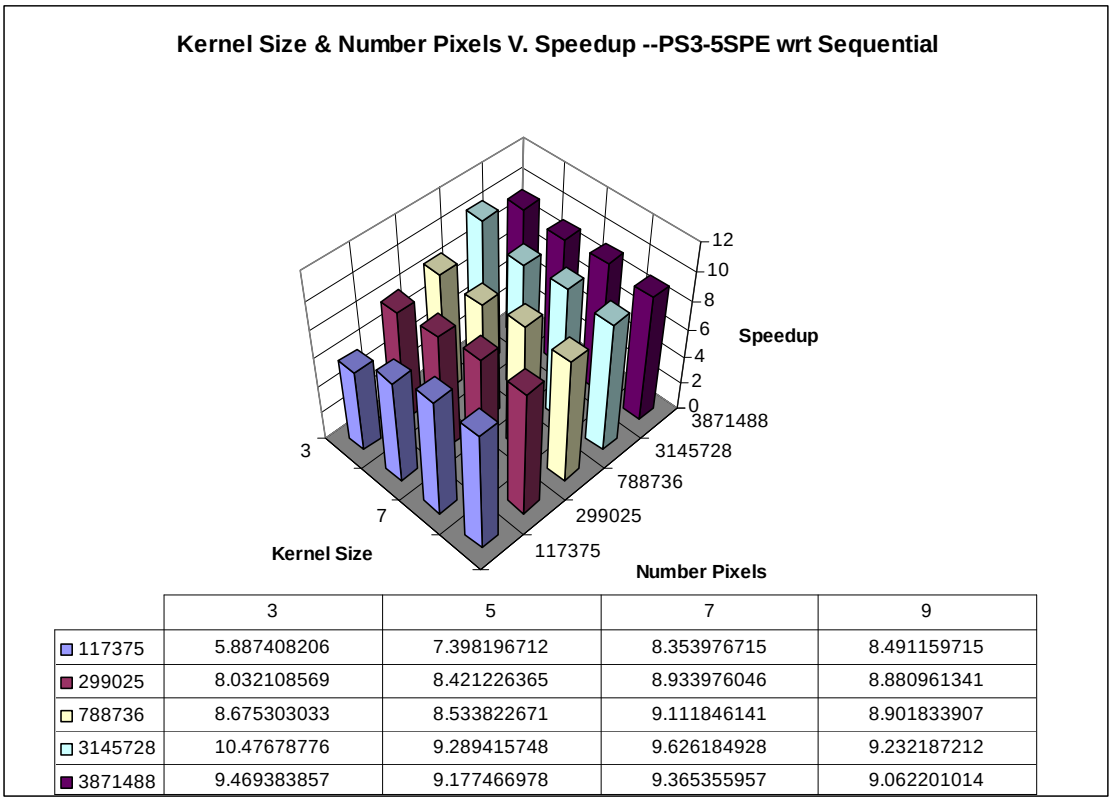
	3	5	7	9
117375	5.460461811	6.219819737	6.926906297	6.880485103
299025	6.820948074	6.864687688	7.28485106	7.180723634
788736	7.105865684	6.90232905	7.444751539	7.238921148
3145728	8.537670103	7.477802707	7.728299687	7.423821281
3871488	7.744359473	7.464198835	7.591442787	7.332070505

Figure 51: Kernel Size & Image Size v. Speedup – PS3-4SPE WRT Sequential



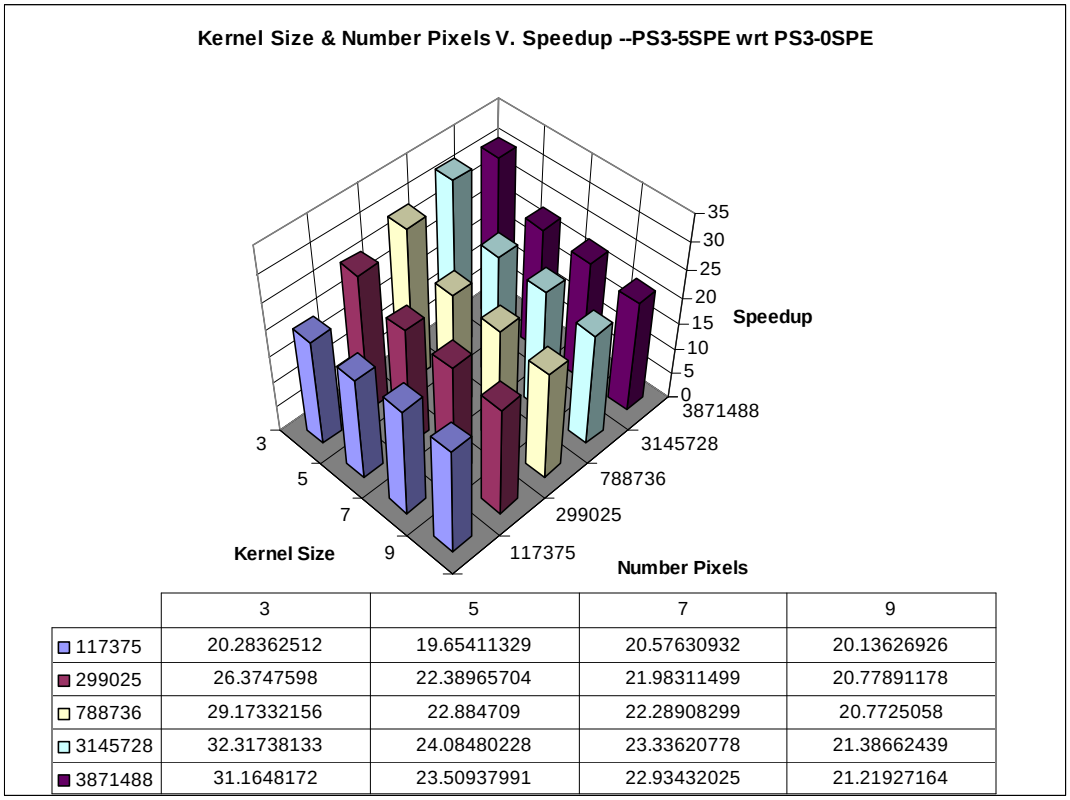
	3	5	7	9
117375	18.81268573	16.52362684	17.0613555	16.3166523
299025	22.39771357	18.25126132	17.92524601	16.80084139
788736	23.89561538	18.50961729	18.21109382	16.89208461
3145728	26.33585281	19.387807	18.73527349	17.19749325
3871488	25.48756614	19.1206012	18.59027898	17.16814662

Figure 52: Kernel Size & Image Size v. Speedup – PS3-4SPE WRT PS3-0SPE



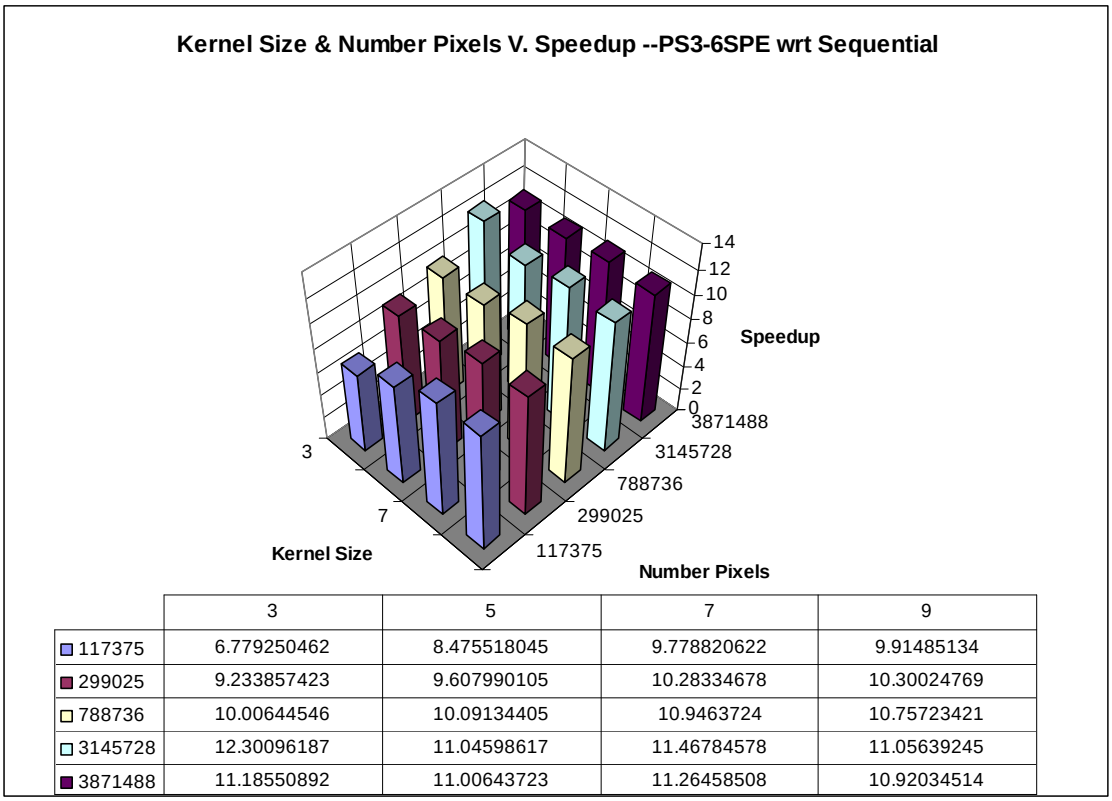
	3	5	7	9
117375	5.887408206	7.398196712	8.353976715	8.491159715
299025	8.032108569	8.421226365	8.933976046	8.880961341
788736	8.675303033	8.533822671	9.111846141	8.901833907
3145728	10.47678776	9.289415748	9.626184928	9.232187212
3871488	9.469383857	9.177466978	9.365355957	9.062201014

Figure 53: Kernel Size & Image Size v. Speedup – PS3-5SPE WRT Sequential



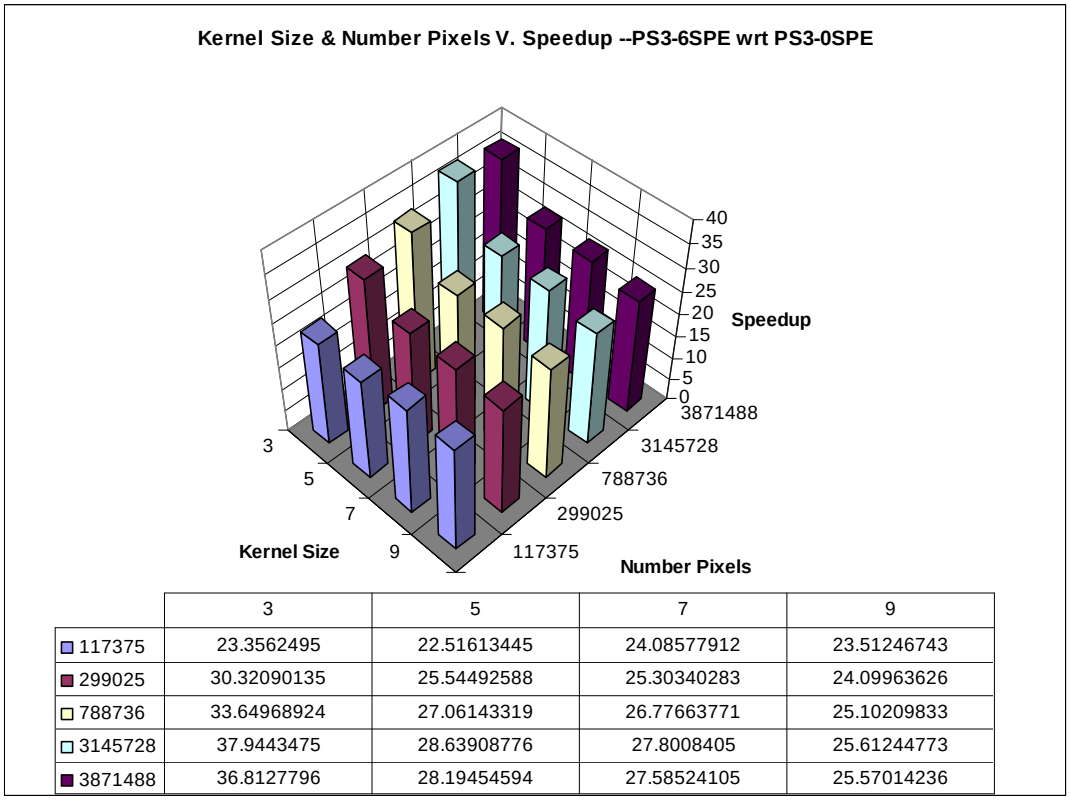
	3	5	7	9
117375	20.28362512	19.65411329	20.57630932	20.13626926
299025	26.3747598	22.38965704	21.98311499	20.77891178
788736	29.17332156	22.884709	22.28908299	20.7725058
3145728	32.31738133	24.08480228	23.33620778	21.38662439
3871488	31.1648172	23.50937991	22.93432025	21.21927164

Figure 54: Kernel Size & Image Size v. Speedup – PS3-5SPE WRT PS3-0SPE



	3	5	7	9
117375	6.779250462	8.475518045	9.778820622	9.91485134
299025	9.233857423	9.607990105	10.28334678	10.30024769
788736	10.00644546	10.09134405	10.9463724	10.75723421
3145728	12.30096187	11.04598617	11.46784578	11.05639245
3871488	11.18550892	11.00643723	11.26458508	10.92034514

Figure 55: Kernel Size & Image Size v. Speedup – PS3-6SPE WRT Sequential



	3	5	7	9
117375	23.3562495	22.51613445	24.08577912	23.51246743
299025	30.32090135	25.54492588	25.30340283	24.09963626
788736	33.64968924	27.06143319	26.77663771	25.10209833
3145728	37.9443475	28.63908776	27.8008405	25.61244773
3871488	36.8127796	28.19454594	27.58524105	25.57014236

Figure 56: Kernel Size & Image Size v. Speedup – PS3-6SPE WRT PS3-0SPE

4.1.6. Test Results – Cell BE: Number of SPEs v. Speedup

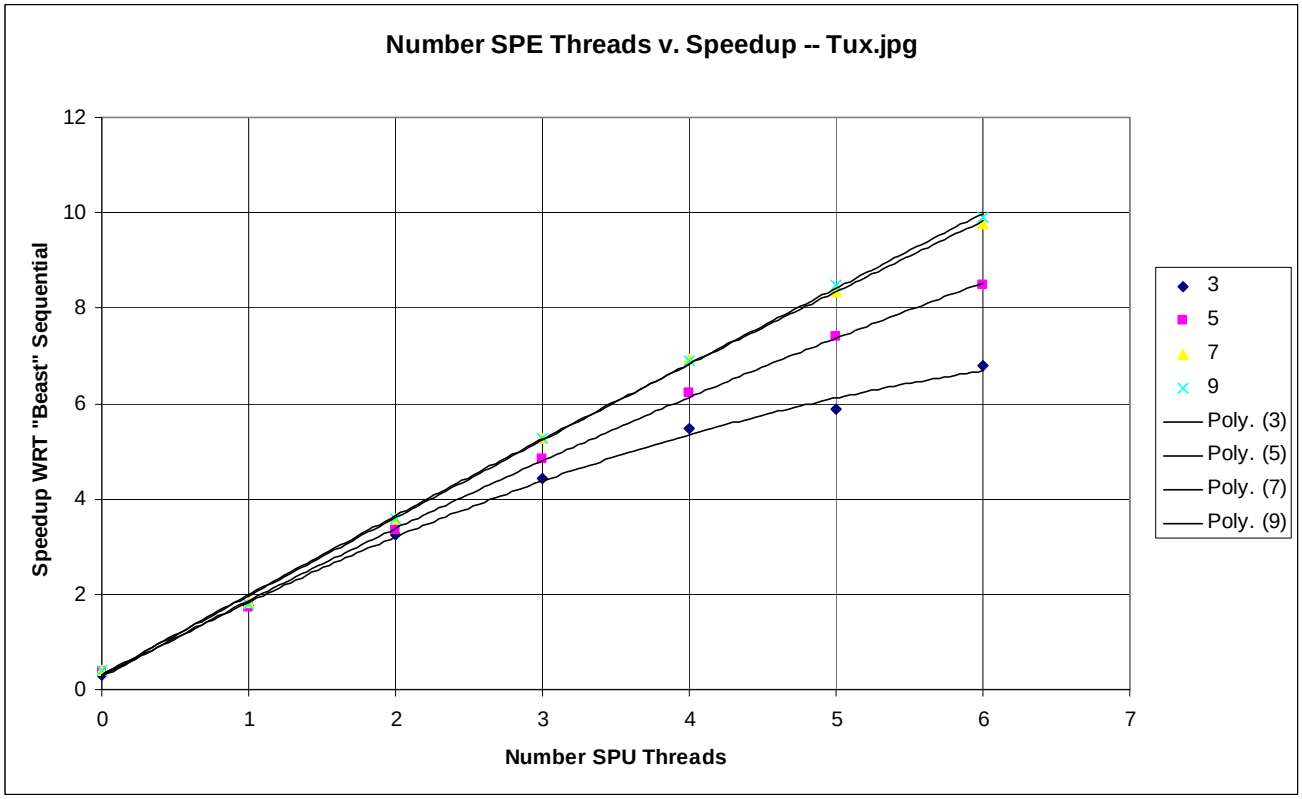


Figure 57: Number of SPEs v. Speedup WRT Sequential – Constant Image: tux.jpg

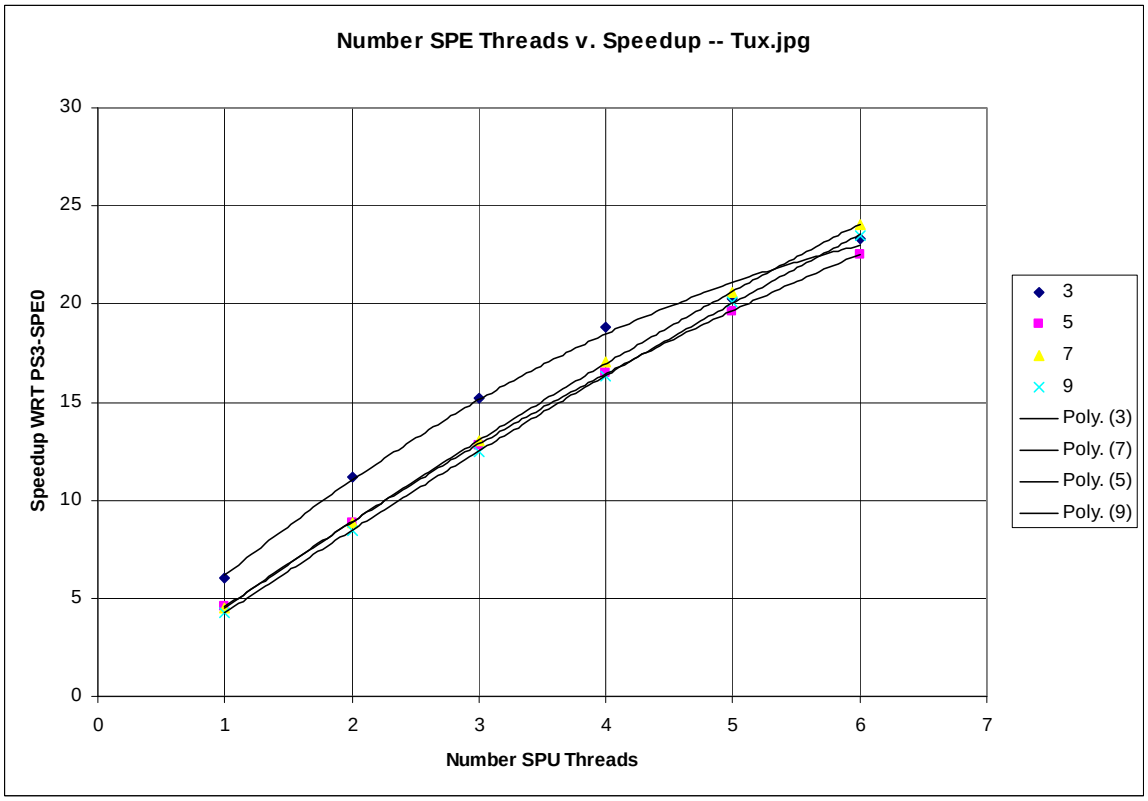


Figure 58: Number of SPEs v. Speedup WRT PS3-0SPE – Constant Image: tux.jpg

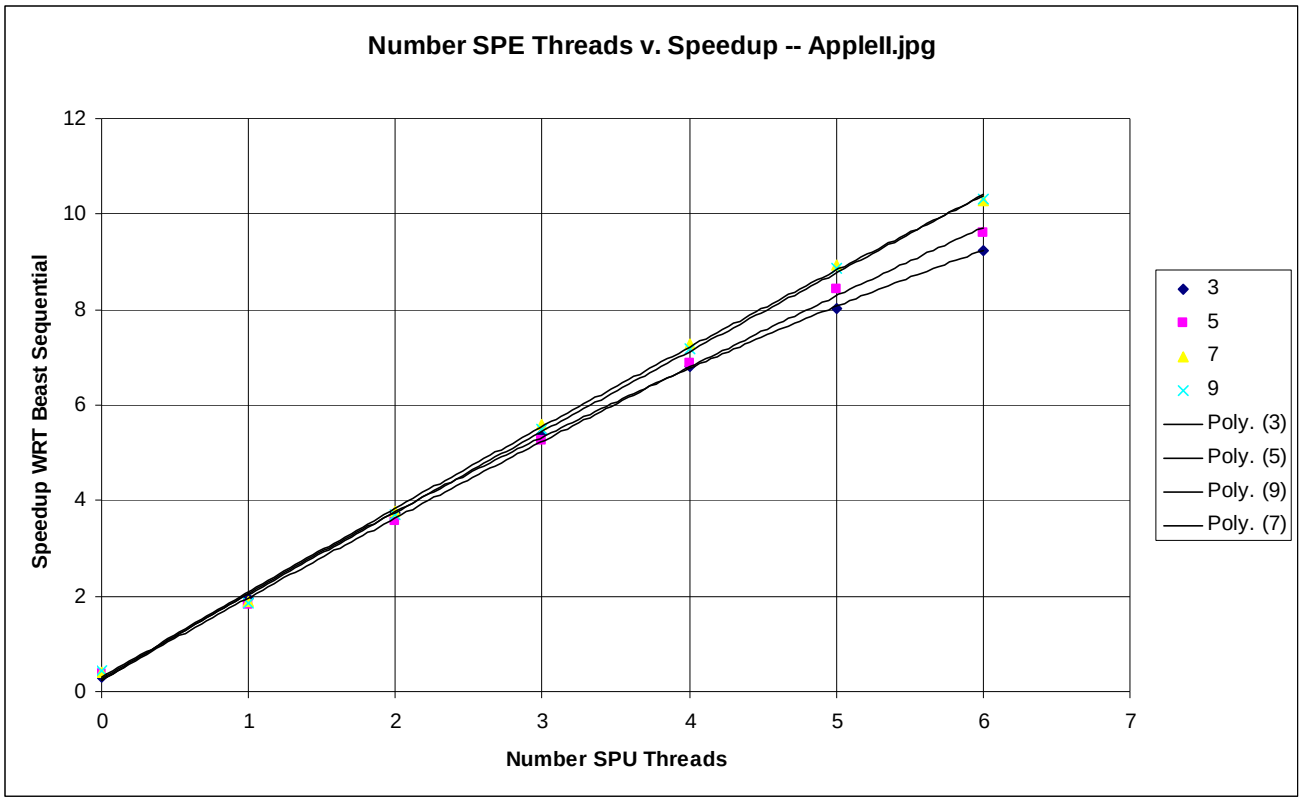


Figure 59: Number of SPEs v. Speedup WRT Sequential – Constant Image: AppleII.jpg

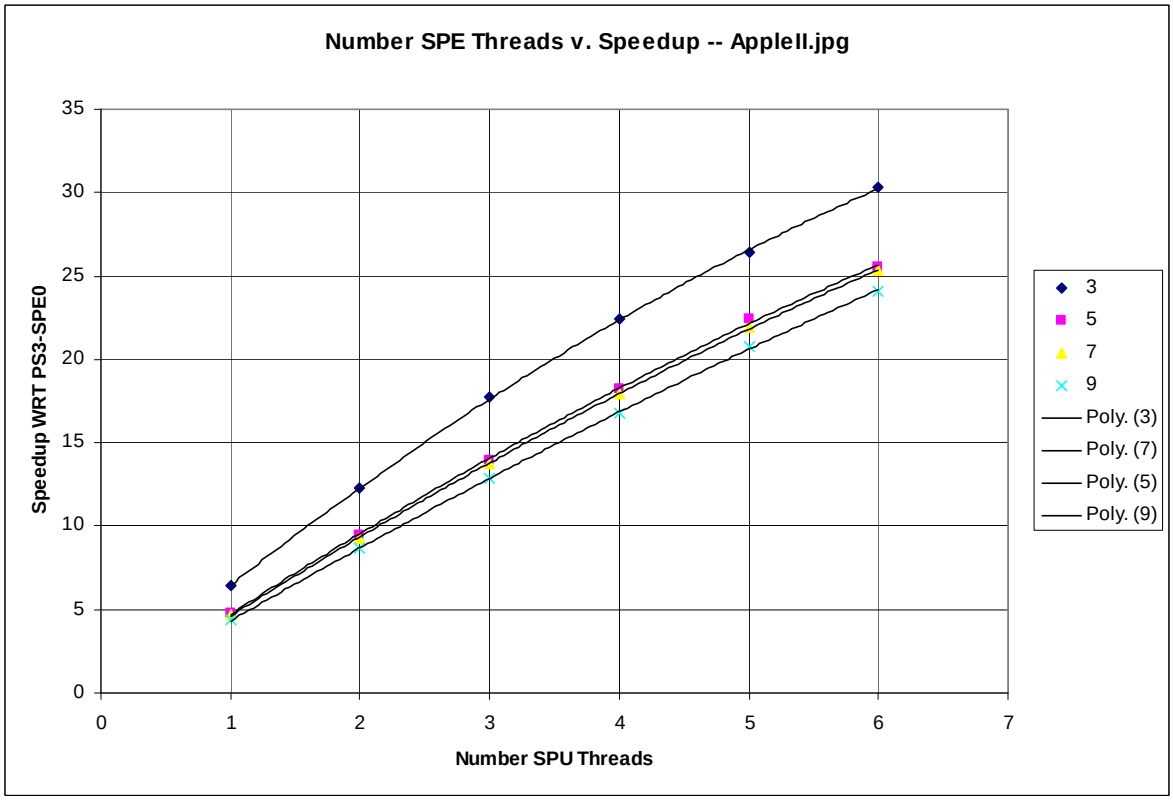


Figure 60: Number of SPEs v. Speedup WRT PS3-0SPE – Constant Image: AppleII.jpg

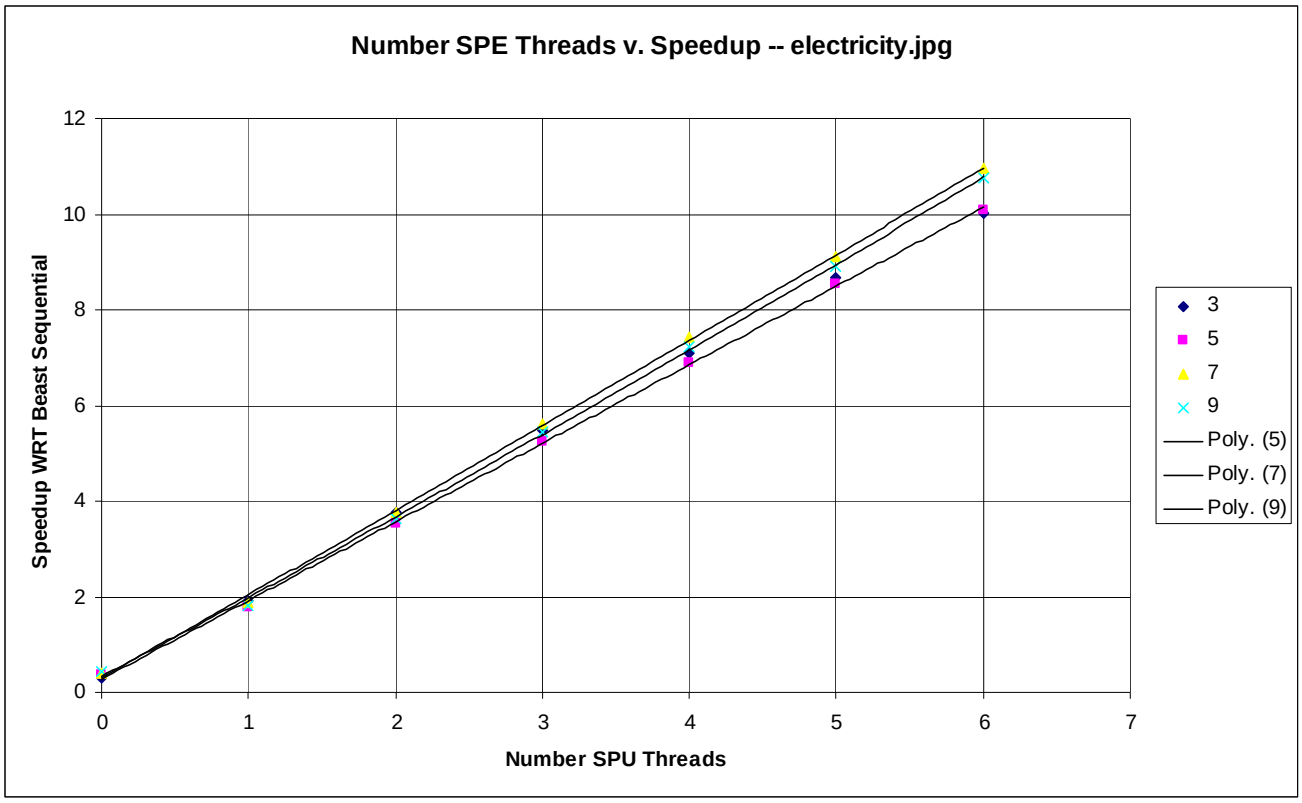


Figure 61: Number of SPEs v. Speedup WRT Sequential – Constant Image: electricity.jpg

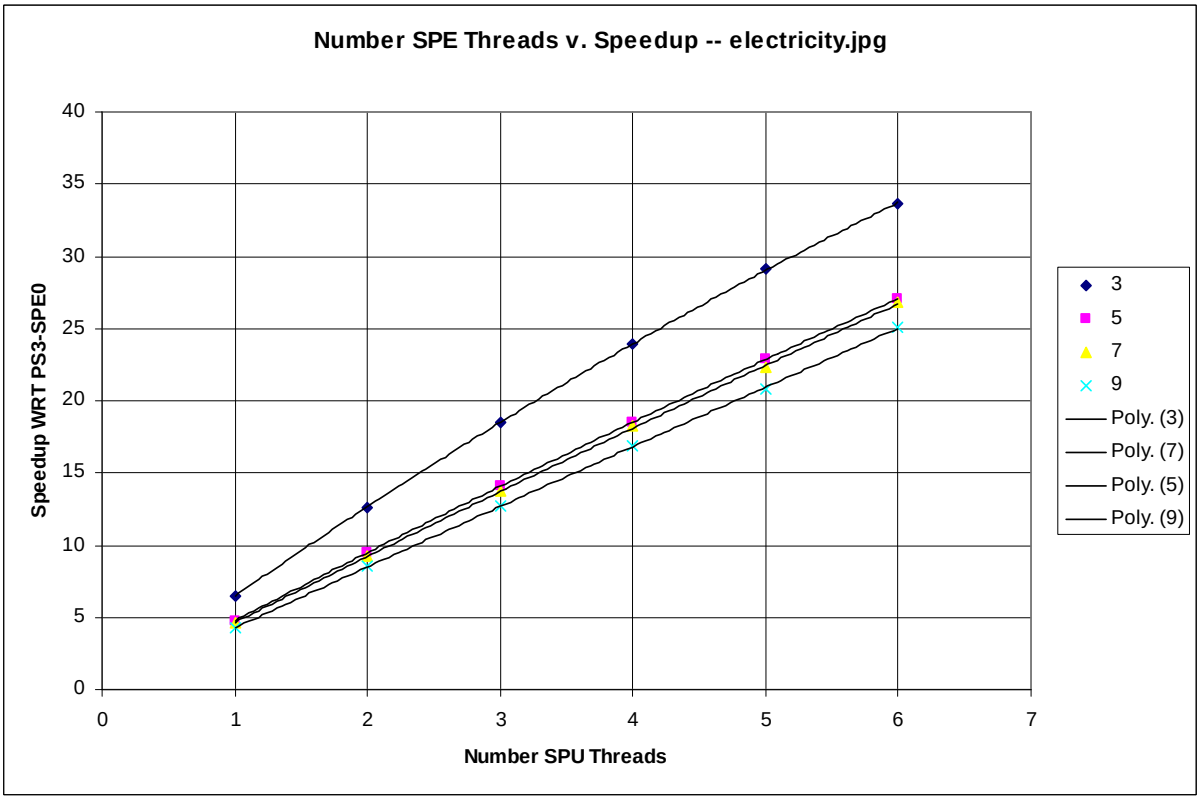


Figure 62: Number of SPEs v. Speedup WRT PS3-0SPE – Constant Image: electricity.jpg

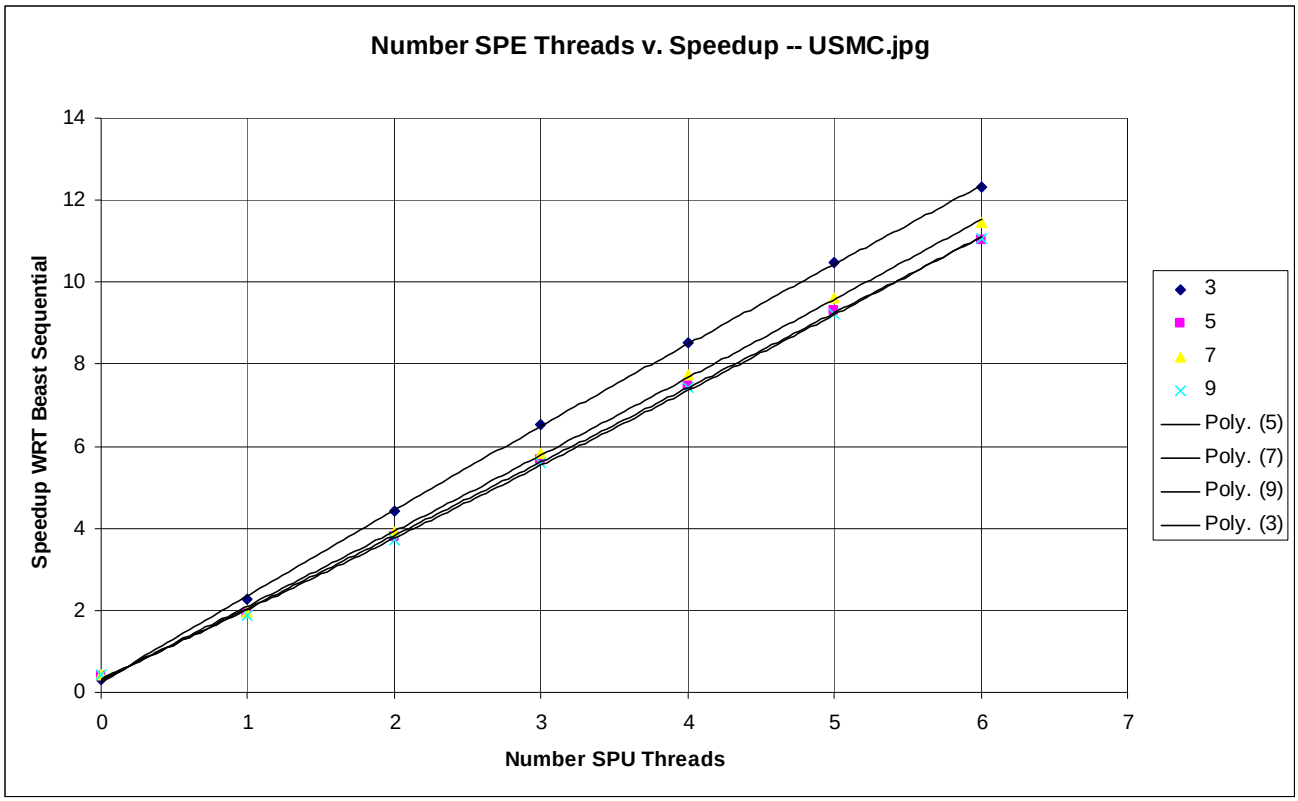


Figure 63: Number of SPEs v. Speedup WRT Sequential – Constant Image: USMC.jpg

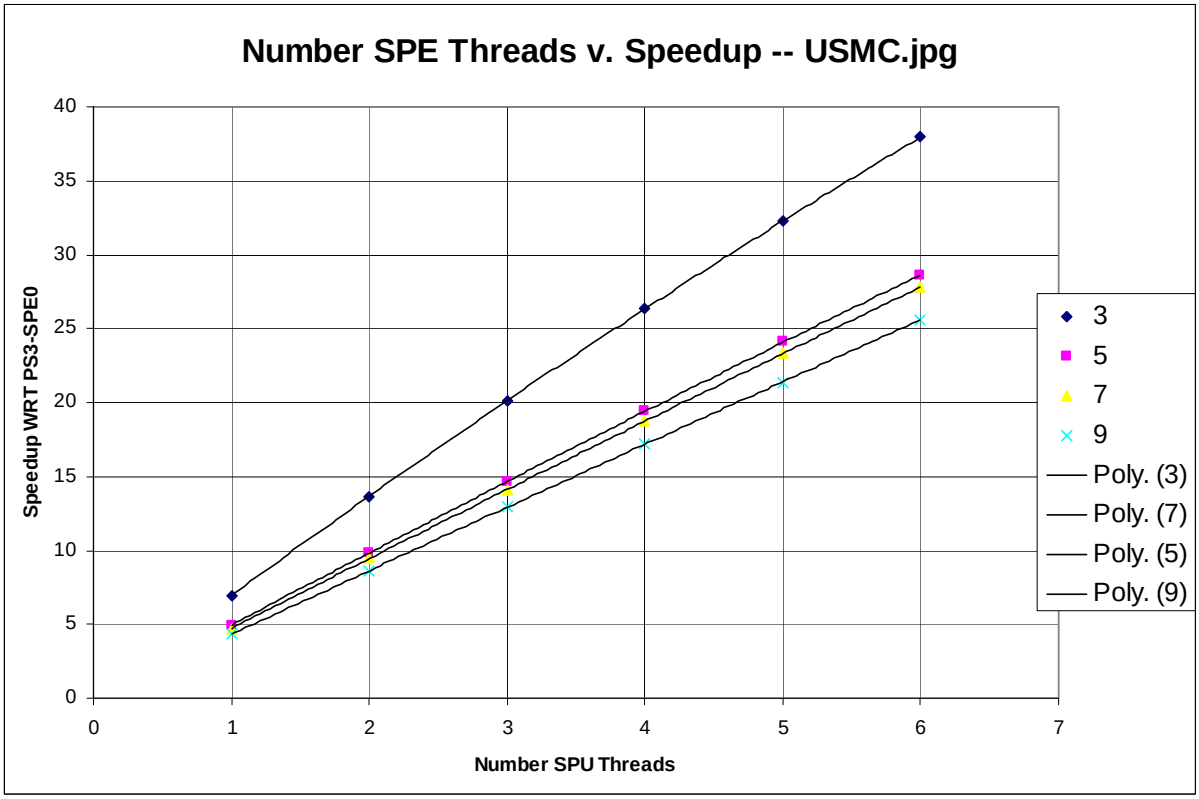


Figure 64: Number of SPEs v. Speedup WRT PS3-0SPE – Constant Image: USMC.jpg

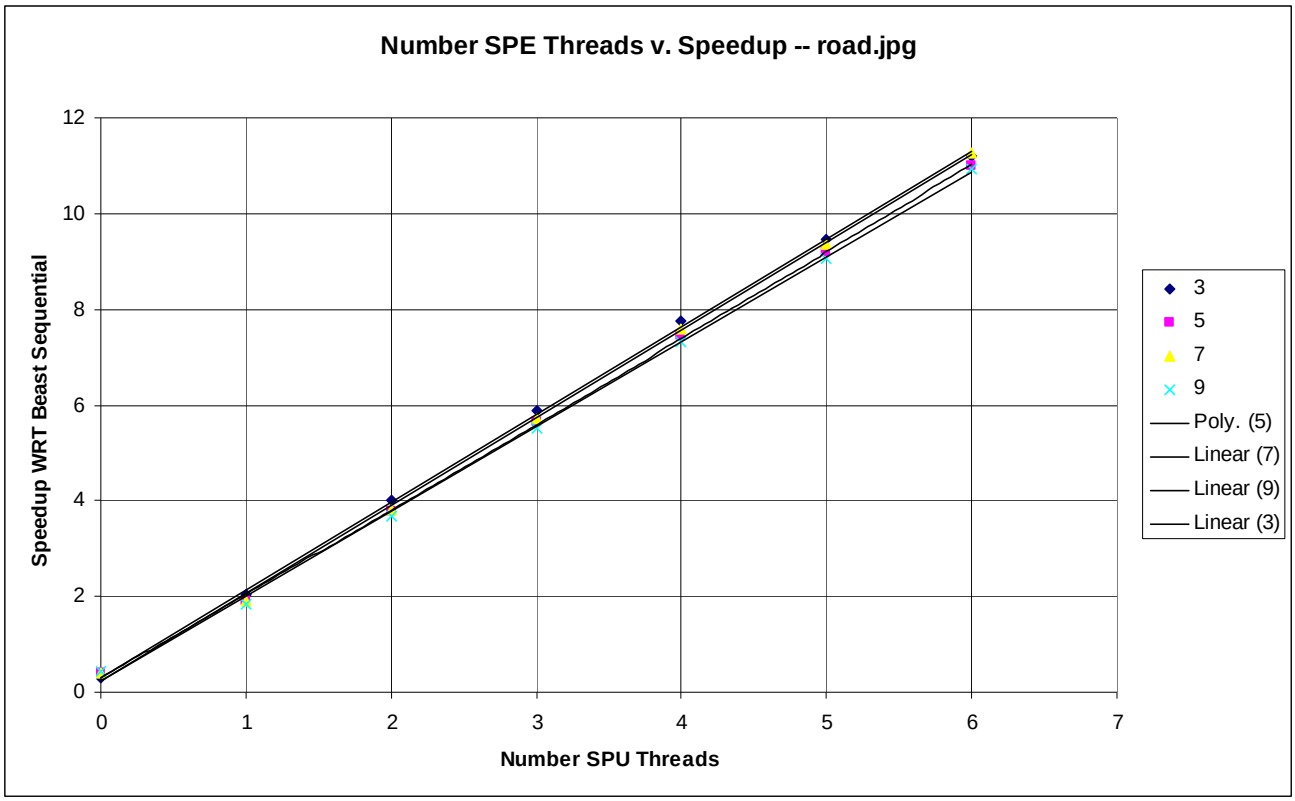


Figure 65: Number of SPEs v. Speedup WRT Sequential – Constant Image: road.jpg

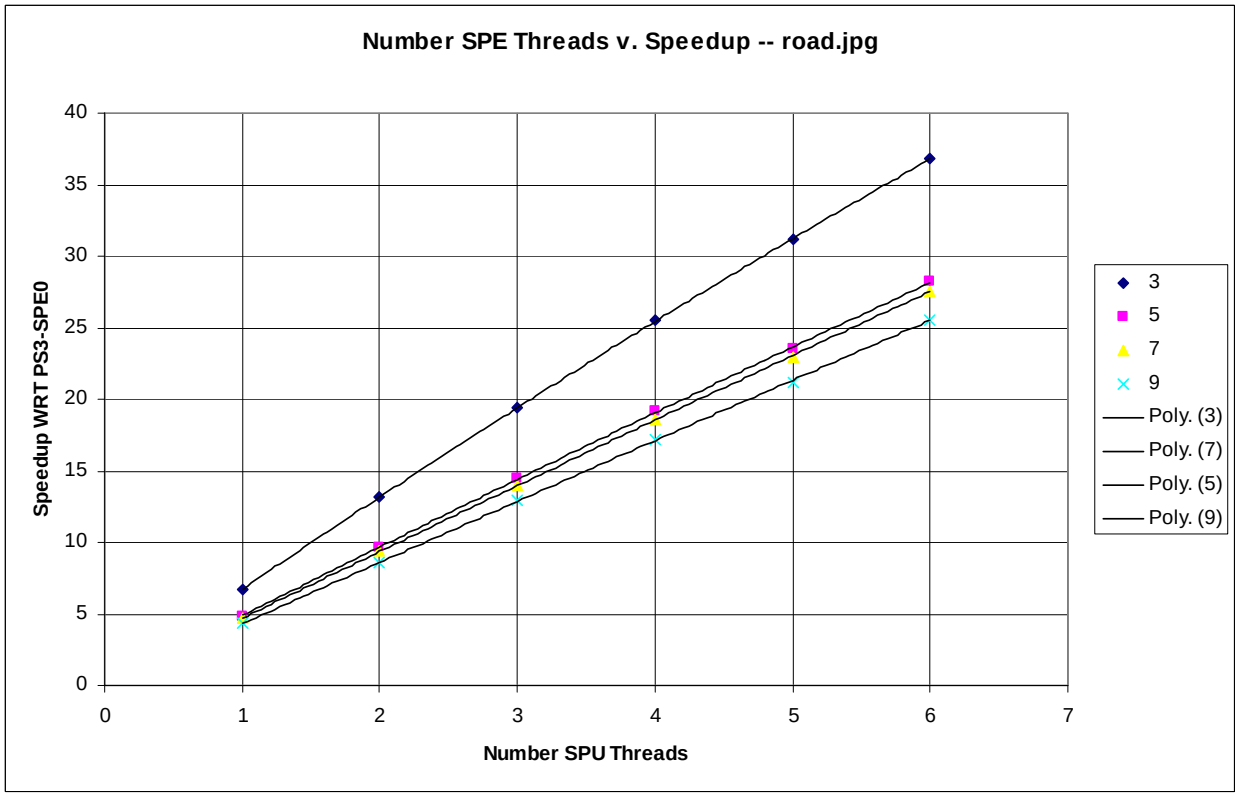


Figure 66: Number of SPEs v. Speedup WRT PS3-0SPE – Constant Image: road.jpg

Figure 67: Number of Multiply Operations v. Time – Linear Scale

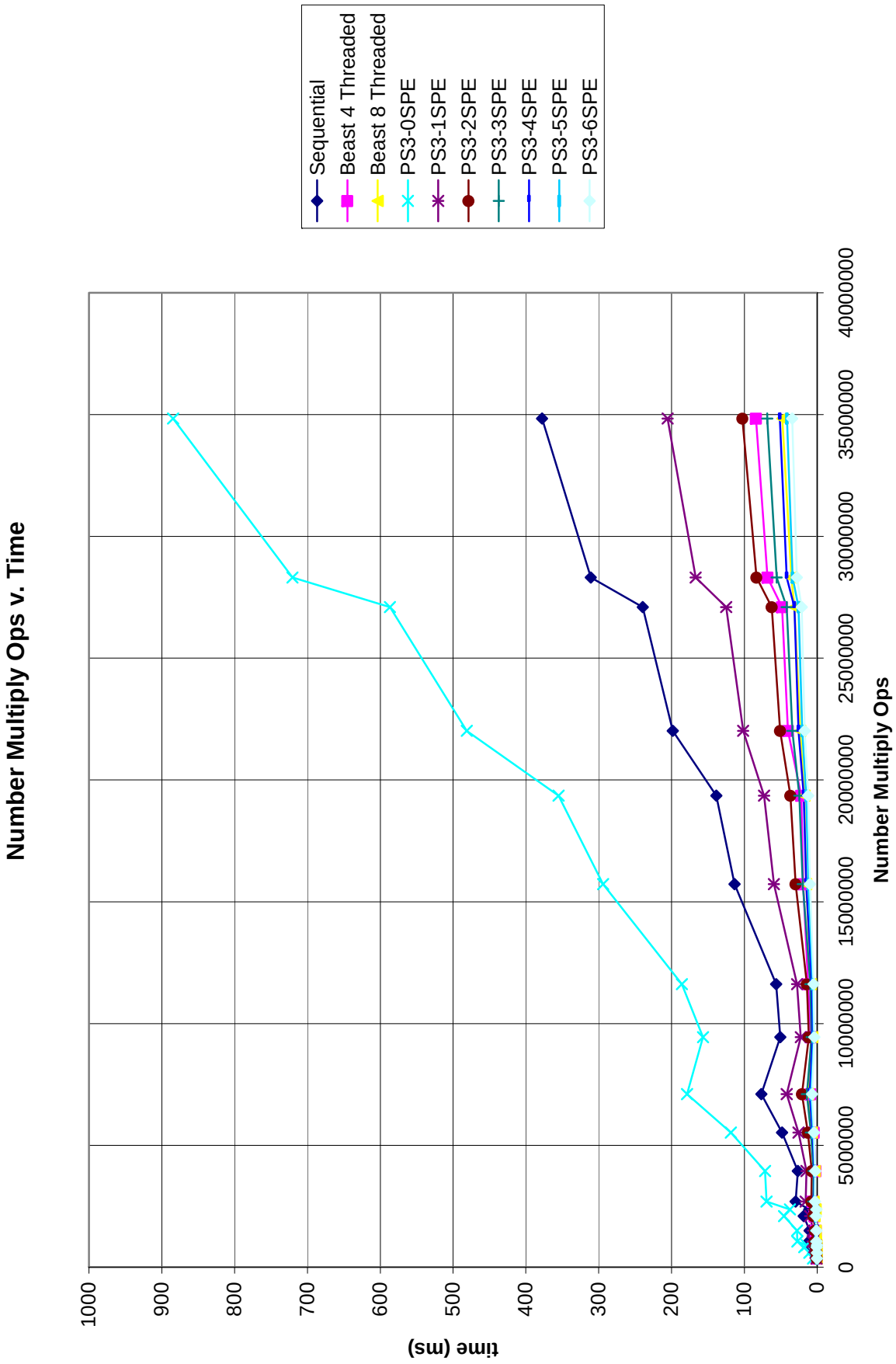


Figure 68: Number of Multiply Operations v. Time – Log Scale

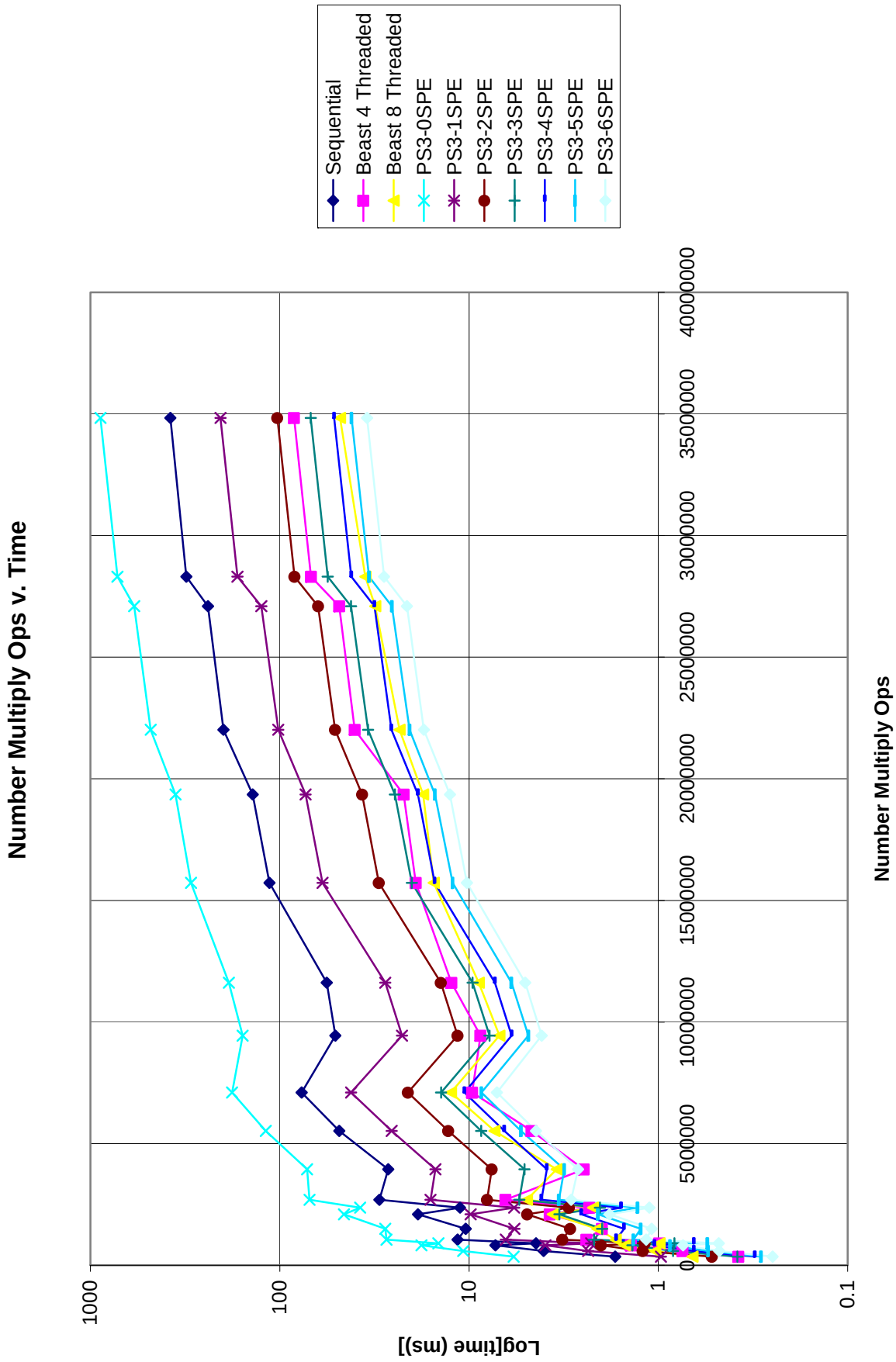


Figure 69: Number of Multiply Operations v. Speedup WRT Sequential Algorithm

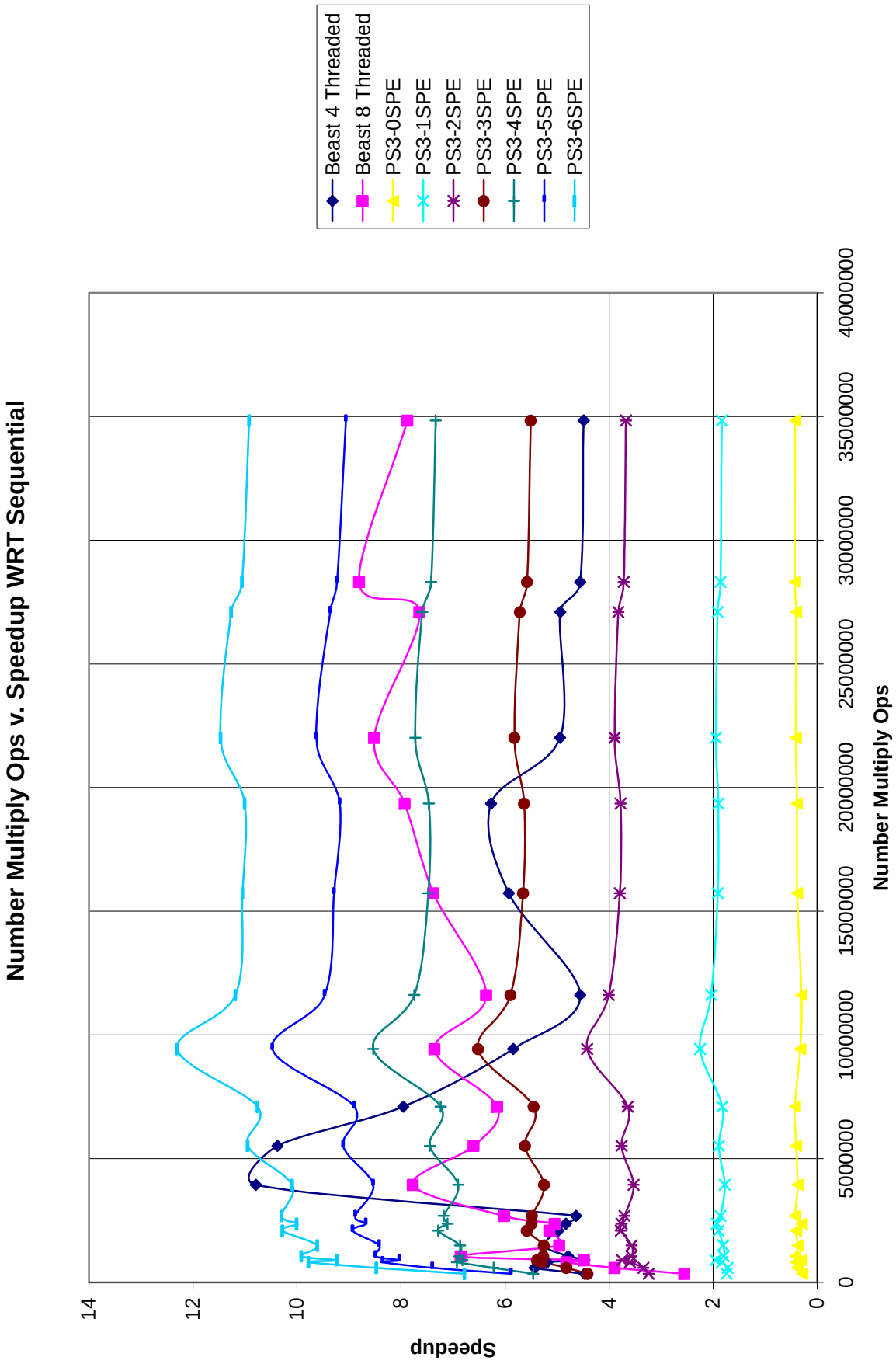
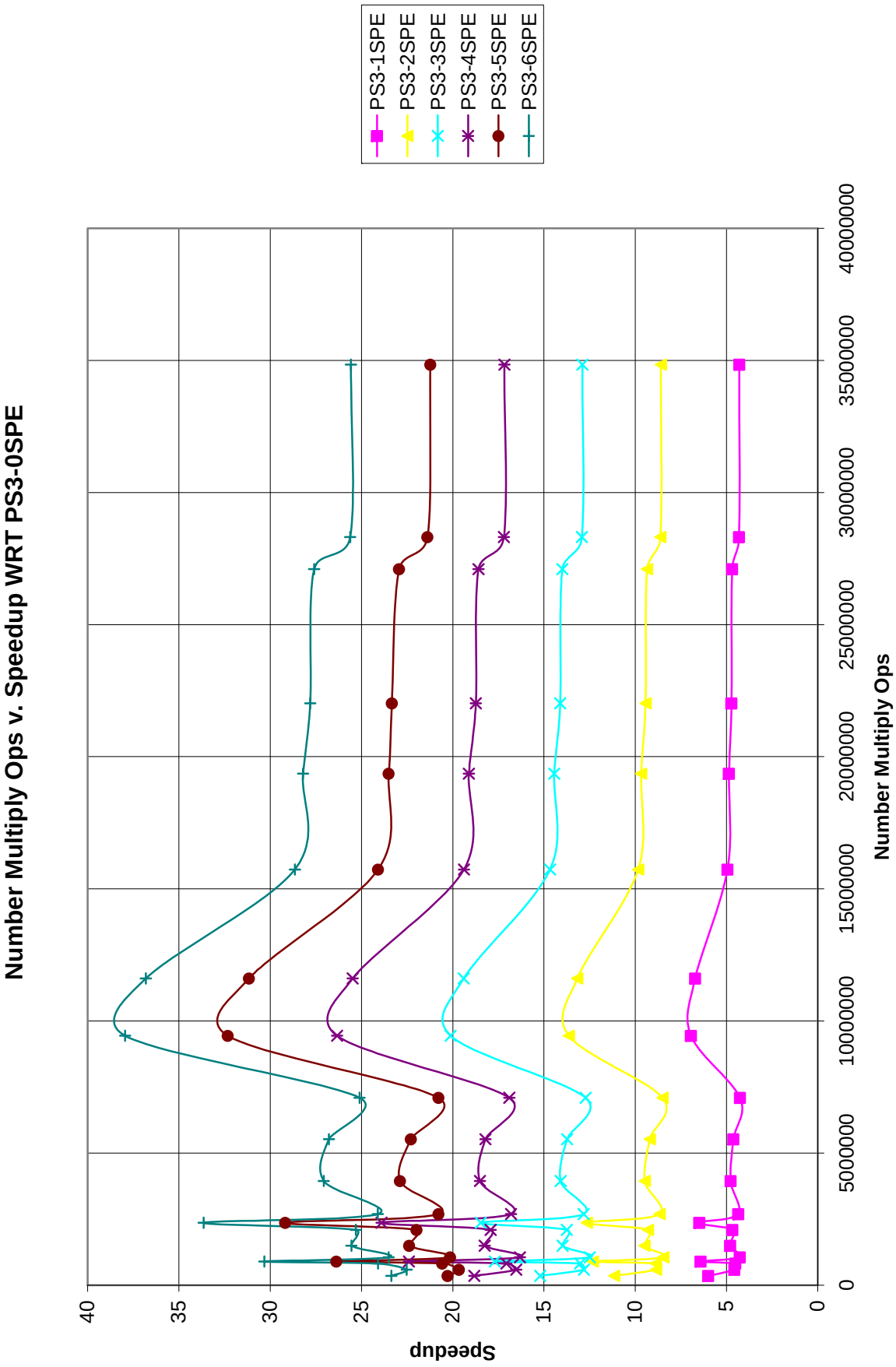


Figure 70: Number of Multiply Operations v. Speedup WRT PS3-0SPE



The above series of graphs indicate that a definite linear relationship between the number of SPEs engaged in the computation and the speedup factor. Additionally, as the image size increases, it begins to dominate the correlation between image and kernel size versus speedup factor. This is consistent with what is know about 2D correlation algorithm as the total number of multiply operations required equals the product of the image size and the kernel. Thus as the image size gets very large and kernel size stays small, in the range 3 to 9, the total execution time becomes dominated by the total number of pixels in the image.

The set of 4 full page plots above that graph total number of multiplication operations (N*k) versus execution time and speedup also agree with the above statements. At smaller total numbers of multiplication operations, the kernel size has a dramatic influence on the speedup and execution time as demonstrated by the dramatic undulations in the left third of the above graphs. However, these undulations become smoother as the total number of operations becomes very large.

With all 6 SPU cores utilized, the PlayStation3 appears to trend towards a speedup factor of approximately 11 as image size becomes large. This is consistent with the claims of IBM that the Cell BE is capable of approximately a 10-fold performance increase over state-of-the-art traditional CPUs.

4.2. Requirements Verification

4.2.1. Requirements Verification Chart

Requirement	Verification
1. The system must achieve an average speedup factor of 2.0 compared to the original single-threaded implementation.	The PlayStation3 has a speedup that depends on image size and kernel size, but as image size increase average speedup tends toward 11 with respect to the Beast Sequential version.
2. The system must use standard libraries, be programmable in C/C++ using GNU compiler/make system.	Entire system is programmed using C/C++ with GNU compiler/make system. See source code in appendix.
3. Total System should cost less than $500.00	Only cost was for single 40GB PlayStation3 which was $398.00 including shipping. All software was free.
4. System should be able to operate in standalone (single machine) mode, or operate over a high speed LAN.	PlayStation3 is running Fedora Core 8 and has full network/internet access via Ethernet. System currently operates in standalone mode, but potentially could be used in a cluster.
5. System should be able to operate from standard AC power mains on less than 400 watts.	According to Stanford's Folding@home, which runs a PS3 distributed network, the PS3 consumes approximately 115 Watts while running on average. Actual power consumption was not measured.
6. System should be able to operate continuously for over 168 hours.	PlayStation3 runs continuously, with brief restarts occasionally required for software updates. No problem experienced. Vastly longer than 168 hours.

Table 4: Requirements Verification Chart

4.3. Standards

4.3.1 Linux/Windows

System uses standard operating systems with very familiar user interfaces. System employs standard Linux and Windows libraries. System follows standard conventions with regard to code readability, comments, and appropriately named files.

5. Summary and Conclusions

5.1. Conclusion

Cell architecture definitely holds a great deal of promise. As evidenced above, properly optimized code designed specifically to take advantage of the unique heterogeneous architecture of the Cell BE can be extremely effective. The SPEs are responsible for the vast majority of the Cell processors power. If the SPEs are not utilized, what is left is essentially an underpowered PowerPC processor. With this particular simple parallel application, the Cell processor found in an inexpensive, sub $400 dollar game system, was able hold its own and even exceed the performance of a multi-thousand dollar desktop with state-of-the-art multi-core x86 processors.

There are several limitations and caveats on the conclusions drawn above. The data presented here for the Sequential, 4 & 8-Threaded implementations was based upon run times on a computer ("The Beast") running full Windows 64 Bit XP. Efforts were made to kill off extraneous processes running in the background but even so, it is by no means a fair comparison to the stripped down version of Linux running in text-only mode on the PlayStation3. With additional time, the first change this author would make is to convert the Sequential, 4 & 8-Threaded programs from the Windows environment to Linux. This conversion is a trivial matter and would eliminate a great deal of the objections to drawing definite conclusions about the relative speedup of the Beast and the PlayStation3.

Besides accomplishing the somewhat abstract and nebulous goal of exploring "Computing in Parallel", the biggest accomplishment of this design project was the setup of a stable development platform for further Cell BE research. With the addition of the PlayStation3 running Linux, OpenCV, and IBM's Cell SDK, the Naval Academy Electrical and Computer Engineering Department has a great new computing resource. The PlayStation3 can even be accessed remotely via SSH for development away from the physical terminal.

An immediate extension of this project would be to actually perform some Cell BE optimization. This requires a much greater understanding of the Cell architecture and specific Cell programming techniques and practices, but is definitely within the realm of possibility now that future students would not have to waste weeks getting the PlayStation3 setup like this project. Future students will be able to immediately jump right into programming the Cell. A great example project would be to optimize a function for OpenCV for the Cell. Although many functions are highly optimized, the open source project is always looking for contributors to optimize functions. The library is far from complete.

Over the course of the year spend on this design project, the author learned several important lessons about the design process. First and foremost, there are always unforeseen problems that eat up time. Initially this project was vastly more ambitious, including a more complex integer factoring algorithm as the benchmark algorithm, and programming a FPGA-based hardware implementation of the algorithms. Additionally, it is vital to always keep backup copies of your work so that you can revert to a previous state when you inevitably mess something up. Additionally, the author regrets not working with at least a partner or a part of a larger group. Although it was great to be able to work on the project whenever it was convenient for the author, it would have been extremely beneficial to be able to divide the workload up amongst at a minimum 2 people. This is the very essence of "computing in parallel".

6. References

[1] Shen, John Paul and Mikko H. Lipasti (2005). *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Professional. p. 561.
[2] Flynn, Laurie J. "Intel Halts Development of 2 New Microprocessors". The New York Times, May 8, 2004.
[3] "Synergistic Processing in Cell's Multicore Architecture".
http://www.research.ibm.com/people/m/mikeg/papers/2006_ieeemicro.pdf. IEEE.

[4] Cell Broadband Engine Programming Handbook. IBM Developer Works. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1741C509C5F64B3300257460006FD68D.
[5] Cell Broadband Engine. http://www-01.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine.

Appendix A: Project Management Plan

A.1. Work Breakdown

ID	Activity	Description	Deliverables/ Checkpoints	Duration	People
1	**Edge Detection/**				
1.1	Research	Research basic edge detection techniques/algorithms and libraries	Pseduocode design for edge detection algorithm	1 month	Nick
1.2	Single-thread, uni-processor	Implement non-threaded, uni-processor version of edge detection algorithm	Running edge detection algorithm	2 weeks	Nick
1.3	Multi-thread, uni-processor	Implement multithreaded, uni-processor version of edge detection algorithm	Running edge detection algorithm	3 weeks	Nick
1.4	Multi-thread, multi-core architecture	Implement multithreaded, multi-processor version of edge detection algorithm	Running edge detection algorithm on “Beast”	3 days	Nick
1.5	Cell BE	Implement Cell based version of edge detection algorithm. Including installation of Linux on PS3	Running edge detection algorithm on Cell processor	2 months	Nick
2	**Data Collection**				
2.1	Data Collection	Data collection and analysis of various implementations	Data Plots	1 week	Nick
3	**Presentation/Report**				
3.1	Final Project Report	Final Project Report	Final report	2 weeks	Nick
3.2	Make Project Poster	Make Project Poster	Rough draft poster	3 days	Nick
3.3	Make PPT Presentation	Make PPT Presentation	Rough draft PPT	2 days	Nick
3.4	Practice for oral presentation	Practice for oral presentation		1 day	Nick
3.5	Presentation	Give oral presentation to EE faculty	PPT presentation Poster	1 day	Nick

A.2. Member Contributions

The entirety of this project was completed by Nicholas Vandal as he is the sole member of the group. Assistance, guidance and advice furnished by Dr. Rakvic, Dr. Ngo, and LT Blair, USN.

A.3. Development Costs

Item	Price
PlayStation3	398.00

Appendix B: Software

B.1. Beast Sequential Code

```
/*
      mainSequential.cpp
      NA Vandal
      April 2009
      USNA EE Final Design PJ

*/

#include "highgui.h"
#include "cv.h"
#include "cxmisc.h"
#include <iostream>

using namespace std;

#define ALIGN128(i) ((i+127)&~127) //16 bytes

//Kernel constants

//Type O
const float valsOrg[] =       {     1.0000, 1.0000, 1.0000,
                                    1.0000, -8.0000, 1.0000,
                                    1.0000, 1.0000, 1.0000};

//Type H
const float sobel3by3_H[] =   {     1, 2, 1,
                                    0, 0, 0,
                                    -1, -2, -1};

//Type V
const float sobel3by3_V[] =   {     1,0,-1,
                                    2,0,-2,
                                    1,0,-1};

//Type P
const float laplacian3by3[] = {     0.1667, 0.6667, 0.1667,
                                    0.6667, -3.3333, 0.6667,
                                    0.1667, 0.667, 0.1667};
//Type L
const float log3by3_025[] =   {     3.5175, 3.5925, 3.5175,
                                    3.5925, -28.4397, 3.5925,
                                    3.5175, 3.5925, 3.5175};

const float log3by3_050[] =   {     0.4038, 0.8021, 0.4038,
                                    0.8021, -4.8233, 0.8021,
                                    0.4038, 0.8021, 0.4038};

const float log3by3_075[] =   {     0.2189, 0.0292, 0.2189,
                                    0.0292, -0.9926, 0.0292,
                                    0.2189, 0.0292, 0.2189};


const float log5by5_025[] =   {     1.2663, 1.2663, 1.2663, 1.2663, 1.2663,
```

```
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.3413, -30.6908, 1.3413, 1.2663,
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.2663, 1.2663, 1.2663, 1.2663};

const float log5by5_050[] =   {     0.0448, 0.0468, 0.0564, 0.0468, 0.0448,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0564, 0.7146, -4.9048, 0.7146, 0.0564,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0448, 0.0468, 0.0564, 0.0468, 0.0448};

const float log5by5_075[] =   {     0.0059, 0.0416, 0.0743, 0.0416, 0.0059,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0743, -0.0451, -1.0058, -0.0451,0.0743,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0059, 0.0416, 0.0743, 0.0416, 0.0059};


const float log7by7_025[] =   {     0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.7211  -31.3110, 0.7211, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461};

const float log7by7_050[] =   {     0.0228, 0.0228, 0.0228, 0.0229, 0.0228,
0.0228, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0229, 0.0345, 0.6927   -4.9267, 0.6927, 0.0345, 0.0229,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0228, 0.0228, 0.0229, 0.0228, 0.0228, 0.0228};

const float log7by7_075[] =   {     0.0001, 0.0002, 0.0011, 0.0024, 0.0011,
0.0002, 0.0001,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0001, 0.0002, 0.0011, 0.0024, 0.0011, 0.0002, 0.0001};

const float log9by9_025[] =   {     0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.4659  -31.5663, 0.4659, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908};
```

```
const float log9by9_050[] =   {     0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0254, 0.6837   -4.9357, 0.6837, 0.0254, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138};

const float log9by9_075[] =   {     0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};


 static CvMat *create_mat( int rows, int cols, int type )
{
    CvMat *ret;
    int is_align = 1; //Alignment required for CVCell only

    if ( is_align ) {
        ret = (CvMat*)cvAlloc( sizeof(CvMat) );
        int step = ALIGN128(cols*CV_ELEM_SIZE(type));
        void *data;
        data = cvAlignPtr(malloc( step * rows + 128 ),128);
        cvInitMatHeader( ret, rows, cols, type, data, step );
    } else {
        ret = cvCreateMat( rows, cols, type );
    }

    /*  touch page */
    memset( ret->data.ptr, 0, ret->step * ret->rows );

    return ret;
}

 static CvMat* getAlignedKernel(char type, int size=3, int coeff = 2)
{
      // *((float *)(kernel->data.ptr + i * kernel->step + j))=vals[c++];

      if(size!=3 && size!=5 && size!=7 && size!=9)
      {
            cout<<"Error: Invalid Kernel Size!\n";
            exit(-1);
      }

      CvMat* kernel;

      if(type=='O')
```

```
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)
			*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=valsOrg[c++];
}
else if(type=='H')
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_H[c++];

}
else if(type=='V')
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_V[c++];

}
else if(type=='P')
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=laplacian3by3[c++];

}
else if(type=='L')
{
	kernel= create_mat(size,size,CV_32FC1);

	if(size==3)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==5)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==7)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }

      }
      else if(size==9)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
```

```
		}
		else
		{
			cout<<"Error: Invalid Kernel Size!\n";
			exit(-1);
		}
	}
	else
	{
		cout<<"Error: Invalid Kernel Type!\n";
		exit(-1);
	}

	return kernel;
}


int main(int argc, char ** argv)
{

	double avgSpeed = 0;
	long numFrame = 0;

	const char* filename = argc >=2 ? argv[1] : "AppleII.jpg";
	double t;

	CvMat * kernel = getAlignedKernel('L',9,2);

	CvMat * kH = getAlignedKernel('H',3,2);
	CvMat * kV = getAlignedKernel('V',3,2);

	IplImage* img = cvLoadImage(filename);
	cvNamedWindow("Example1",CV_WINDOW_AUTOSIZE);

	IplImage* gray = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
	IplImage* out = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);

	IplImage* out2 = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);

	cvCvtColor(img,gray,CV_BGR2GRAY);


	while(1)
	{
		t = (double)cvGetTickCount();

		cvFilter2D(gray,out,kernel);

		//cvFilter2D(gray,out,kV);
		//cvFilter2D(gray,out2,kH);
		//cvAdd(out,out2,out);

		t = (double)cvGetTickCount() - t;

		printf( "detection time = %gms\n",
		t/((double)cvGetTickFrequency()*1000.) );
```

```
            //Average speed
             avgSpeed +=t;
             numFrame++;
            printf("Average = %f\n",
            (avgSpeed/(double)numFrame)/(((double)cvGetTickFrequency()*1000.)
            ));

             if(numFrame==100)
                  break;

      }

      //cvThreshold(out,out,20,255,CV_THRESH_BINARY);
      //cvThreshold(out,out,100,255,CV_THRESH_BINARY);
      cvShowImage("Example1",out);
      cvSaveImage("outSeq.jpg",out );
      cvThreshold(out,out,75,255,CV_THRESH_BINARY);
      cvSaveImage("outSeq_thres.jpg",out );


      cvWaitKey(0);
      cvReleaseImage(&img);
      cvReleaseImage(&gray);
      cvReleaseImage(&out);
      cvReleaseImage(&out2);
      cvReleaseMat(&kernel);
      cvReleaseMat(&kV);
      cvReleaseMat(&kH);
      cvDestroyWindow("Example1");

      return 0;


}
```

B.2. Beast 4 Threaded Code

```
/*
      main4Threaded.cpp
      NA Vandal
      April 2009
      USNA EE Final Design PJ

*/

#include "highgui.h"
#include "cv.h"
#include <iostream>
#include <process.h>
#include <windows.h>
#include "cxmisc.h"

#define IMAGE_DATA_PTR(origIm, x, y) ((origIm)->imageData + (y)*(origIm)-
>widthStep + (x)*(origIm)->nChannels)

using namespace std;

CvMat* kernel;

IplImage* g_output = NULL;
IplImage* g_output_whole = NULL;
IplImage* g_gray = NULL;

//Sub image src headers
IplImage *sub_imgTopLeft_Src =NULL;
IplImage *sub_imgTopRight_Src  =NULL;
IplImage *sub_imgBottomLeft_Src =NULL;
IplImage *sub_imgBottomRight_Src  =NULL;

//Sub image dst headers
IplImage *sub_imgTopLeft_Dst =NULL;
IplImage *sub_imgTopRight_Dst =NULL;
IplImage *sub_imgBottomLeft_Dst=NULL;
IplImage *sub_imgBottomRight_Dst =NULL;

//Thread handles
const int THREADCOUNT = 4;
HANDLE aThread[THREADCOUNT];
HANDLE hWorkerThreadDone[THREADCOUNT];
HANDLE hEventMoreWorkToDo[THREADCOUNT];

#define ALIGN128(i) ((i+127)&~127) //16 bytes

static CvMat *create_mat( int rows, int cols, int type )
{
    CvMat *ret;
    int is_align = 1;

    if ( is_align ) {
        ret = (CvMat*)cvAlloc( sizeof(CvMat) );
        int step = ALIGN128(cols*CV_ELEM_SIZE(type));
        void *data;
```

```
        data = cvAlignPtr(malloc( step * rows + 128 ),128);
        cvInitMatHeader( ret, rows, cols, type, data, step );
    } else {
        ret = cvCreateMat( rows, cols, type );
    }

    /*  touch page */
    memset( ret->data.ptr, 0, ret->step * ret->rows );

    return ret;
}
//Kernel constants
//Type O
const float valsOrg[] =       {     1.0000, 1.0000, 1.0000,
                                    1.0000, -8.0000, 1.0000,
                                    1.0000, 1.0000, 1.0000};

//Type H
const float sobel3by3_H[] =   {     1, 2, 1,
                                    0, 0, 0,
                                    -1, -2, -1};

//Type V
const float sobel3by3_V[] =   {     1,0,-1,
                                    2,0,-2,
                                    1,0,-1};

//Type P
const float laplacian3by3[] = {     0.1667, 0.6667, 0.1667,
                                    0.6667, -3.3333, 0.6667,
                                    0.1667, 0.667, 0.1667};
//Type L
const float log3by3_025[] =   {     3.5175, 3.5925, 3.5175,
                                    3.5925, -28.4397, 3.5925,
                                    3.5175, 3.5925, 3.5175};

const float log3by3_050[] =   {     0.4038, 0.8021, 0.4038,
                                    0.8021, -4.8233, 0.8021,
                                    0.4038, 0.8021, 0.4038};

const float log3by3_075[] =   {     0.2189, 0.0292, 0.2189,
                                    0.0292, -0.9926, 0.0292,
                                    0.2189, 0.0292, 0.2189};


const float log5by5_025[] =   {     1.2663, 1.2663, 1.2663, 1.2663, 1.2663,
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.3413, -30.6908, 1.3413, 1.2663,
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.2663, 1.2663, 1.2663, 1.2663};

const float log5by5_050[] =   {     0.0448, 0.0468, 0.0564, 0.0468, 0.0448,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0564, 0.7146, -4.9048, 0.7146, 0.0564,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0448, 0.0468, 0.0564, 0.0468, 0.0448};
```

```
const float log5by5_075[] =   {     0.0059, 0.0416, 0.0743, 0.0416, 0.0059,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0743, -0.0451, -1.0058, -0.0451,0.0743,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0059, 0.0416, 0.0743, 0.0416, 0.0059};


const float log7by7_025[] =   {     0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.7211  -31.3110, 0.7211, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461};

const float log7by7_050[] =   {     0.0228, 0.0228, 0.0228, 0.0229, 0.0228,
0.0228, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0229, 0.0345, 0.6927   -4.9267, 0.6927, 0.0345, 0.0229,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0228, 0.0228, 0.0229, 0.0228, 0.0228, 0.0228};

const float log7by7_075[] =   {     0.0001, 0.0002, 0.0011, 0.0024, 0.0011,
0.0002, 0.0001,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0001, 0.0002, 0.0011, 0.0024, 0.0011, 0.0002, 0.0001};

const float log9by9_025[] =   {     0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.4659  -31.5663, 0.4659, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908};

const float log9by9_050[] =   {     0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0254, 0.6837   -4.9357, 0.6837, 0.0254, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138};
```

```
const float log9by9_075[] =   {     0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};

static CvMat* getAlignedKernel(char type, int size=3, int coeff = 2)
{
      // *((float *)(kernel->data.ptr + i * kernel->step + j))=vals[c++];

      if(size!=3 && size!=5 && size!=7 && size!=9)
      {
            cout<<"Error: Invalid Kernel Size!\n";
            exit(-1);
      }

      CvMat* kernel;

      if(type=='O')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)
                        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=valsOrg[c++];
      }
      else if(type=='H')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)

      *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_H[c++];

      }
      else if(type=='V')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)

      *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_V[c++];

      }
      else if(type=='P')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=laplacian3by3[c++];

}
else if(type=='L')
{
	kernel= create_mat(size,size,CV_32FC1);

	if(size==3)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_050[c++];
		}
		else if(coeff==3)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_075[c++];
		}
		else
		{
			cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
			exit(-1);
		}
	}
	else if(size==5)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_050[c++];
```

```
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==7)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }

      }
      else if(size==9)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_025[c++];
```

```
                }
                else if(coeff==2)
                {
                        int c=0;
                        for(int i=0; i<size; ++i)
                                for(int j=0; j<size; ++j)

        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_050[c++];
                }
                else if(coeff==3)
                {
                        int c=0;
                        for(int i=0; i<size; ++i)
                                for(int j=0; j<size; ++j)

        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_075[c++];
                }
                else
                {
                        cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                        exit(-1);
                }
        }
        else
        {
                cout<<"Error: Invalid Kernel Size!\n";
                exit(-1);
        }
    }
    else
    {
        cout<<"Error: Invalid Kernel Type!\n";
        exit(-1);
    }



    return kernel;
}
unsigned __stdcall helperThreadFunc( void* pArguments )

{
    //myData* a = (myData*)pArguments;
    int q = (int)pArguments;
    DWORD dwRet;

    //printf("In worker thread %i...\n",q);
    do{
        //printf("Worker thread %i processing...\n",q);
        //Do  work on specific quadrant based upon argument
        if(q == 0) //Top left
        {
            cvFilter2D(sub_imgTopLeft_Src,sub_imgTopLeft_Dst,kernel);
        }
        else if(q ==1) //Top right
        {
```

```
                cvFilter2D(sub_imgTopRight_Src,sub_imgTopRight_Dst,kernel);
            }

            else if(q ==2) //Bottom left
            {

                cvFilter2D(sub_imgBottomRight_Src,sub_imgBottomRight_Dst,ke
                rnel);


            }
            else if(q ==3) //Bottom right
            {

                cvFilter2D(sub_imgBottomLeft_Src,sub_imgBottomLeft_Dst,kern
                el);
            }



            //Signal that has completed work, wait for moreWork event from
            //main thread
            if (! SetEvent(hWorkerThreadDone[q]))
            {
                printf("SetEvent failed (%d)\n", GetLastError());

                _endthreadex( 0 );
                return 0;
            }

            //printf("Worker thread %i waiting for signal to resume
            //processing...\n",q);
            dwRet = WaitForSingleObject(hEventMoreWorkToDo[q],INFINITE);

      }while(dwRet == WAIT_OBJECT_0);

    _endthreadex( 0 );

      return 0;

}

int main(int argc, char ** argv)
{
      double avgSpeed = 0;
      long numFrame = 0;

   const char* filename = argc >=2 ? argv[1] : "USMC.jpg";
   double t;

   //Create kernel
      kernel = getAlignedKernel('L',9,2);

      IplImage* img = cvLoadImage(filename);

      g_gray = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
      g_output = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
```

```
	cvCvtColor(img,g_gray,CV_BGR2GRAY);

	unsigned TC;

	if(!img) return -1;

	//Filter frame
	cvCvtColor(img,g_gray,CV_RGB2GRAY);

	//Time thread creation/overhead
	 t = (double)cvGetTickCount();

	//***Setup sub regions --- Source
**********************************************************

	//Top Left
	sub_imgTopLeft_Src = cvCreateImageHeader(
		cvSize(g_gray->width/2,g_gray->height/2),
		g_gray->depth,
		g_gray->nChannels
		);
	sub_imgTopLeft_Src ->origin = g_gray->origin;
	sub_imgTopLeft_Src ->widthStep=g_gray->widthStep;
	sub_imgTopLeft_Src ->imageData = IMAGE_DATA_PTR(g_gray, 0, 0);
	sub_imgTopLeft_Src ->imageDataOrigin = g_gray->imageDataOrigin;

	//Top Right
	sub_imgTopRight_Src = cvCreateImageHeader(
		cvSize(g_gray->width/2,g_gray->height/2),
		g_gray->depth,
		g_gray->nChannels
		);
	sub_imgTopRight_Src ->origin = g_gray->origin;
	sub_imgTopRight_Src ->widthStep=g_gray->widthStep;
	sub_imgTopRight_Src ->imageData = IMAGE_DATA_PTR(g_gray, g_gray-
>width/2, 0);
	sub_imgTopRight_Src ->imageDataOrigin = g_gray->imageDataOrigin;

	//Bottom left
	sub_imgBottomLeft_Src = cvCreateImageHeader(
		cvSize(g_gray->width/2,g_gray->height/2),
		g_gray->depth,
		g_gray->nChannels
		);
	sub_imgBottomLeft_Src->origin = g_gray->origin;
	sub_imgBottomLeft_Src->widthStep=g_gray->widthStep;
	sub_imgBottomLeft_Src->imageData = IMAGE_DATA_PTR(g_gray, 0, g_gray-
>height/2);
	sub_imgBottomLeft_Src->imageDataOrigin = g_gray->imageDataOrigin;

	//Bottom right
	sub_imgBottomRight_Src = cvCreateImageHeader(
		cvSize(g_gray->width/2,g_gray->height/2),
		g_gray->depth,
		g_gray->nChannels
		);
```

```
	sub_imgBottomRight_Src->origin = g_gray->origin;
	sub_imgBottomRight_Src->widthStep=g_gray->widthStep;
	sub_imgBottomRight_Src->imageData = IMAGE_DATA_PTR(g_gray, g_gray-
>width/2, g_gray->height/2);
	sub_imgBottomRight_Src->imageDataOrigin = g_gray->imageDataOrigin;

	//***Setup sub regions --- Destination
************************************************

	//Top Left
	sub_imgTopLeft_Dst = cvCreateImageHeader(
		cvSize(g_output->width/2,g_output->height/2),
		g_output->depth,
		g_output->nChannels
		);
	sub_imgTopLeft_Dst ->origin = g_output->origin;
	sub_imgTopLeft_Dst ->widthStep=g_output->widthStep;
	sub_imgTopLeft_Dst ->imageData = IMAGE_DATA_PTR(g_output, 0, 0);
	sub_imgTopLeft_Dst ->imageDataOrigin = g_output->imageDataOrigin;

	//Top Right
	sub_imgTopRight_Dst = cvCreateImageHeader(
		cvSize(g_output->width/2,g_output->height/2),
		g_output->depth,
		g_output->nChannels
		);
	sub_imgTopRight_Dst ->origin = g_output->origin;
	sub_imgTopRight_Dst ->widthStep=g_output->widthStep;
	sub_imgTopRight_Dst ->imageData = IMAGE_DATA_PTR(g_output, g_output-
>width/2, 0);
	sub_imgTopRight_Dst ->imageDataOrigin = g_output->imageDataOrigin;

	//Bottom left
	sub_imgBottomLeft_Dst = cvCreateImageHeader(
		cvSize(g_output->width/2,g_output->height/2),
		g_output->depth,
		g_output->nChannels
		);
	sub_imgBottomLeft_Dst->origin = g_output->origin;
	sub_imgBottomLeft_Dst->widthStep=g_output->widthStep;
	sub_imgBottomLeft_Dst->imageData = IMAGE_DATA_PTR(g_output, 0,
g_output->height/2);
	sub_imgBottomLeft_Dst->imageDataOrigin = g_output->imageDataOrigin;

	//Bottom right
	sub_imgBottomRight_Dst = cvCreateImageHeader(
		cvSize(g_output->width/2,g_output->height/2),
		g_output->depth,
		g_output->nChannels
		);
	sub_imgBottomRight_Dst->origin = g_output->origin;
	sub_imgBottomRight_Dst->widthStep=g_output->widthStep;
	sub_imgBottomRight_Dst->imageData = IMAGE_DATA_PTR(g_output, g_output-
>width/2, g_output->height/2);
	sub_imgBottomRight_Dst->imageDataOrigin = g_output->imageDataOrigin;
```

```
      // Create worker threads
    for( int i=0; i < THREADCOUNT; i++ )
    {
            //Create worker thread
        aThread[i] = (HANDLE)_beginthreadex(
                     NULL,       // default security attributes
                     0,          // default stack size
                     &helperThreadFunc,
                     (void*)i,       //thread function arguments
                     0,          // default creation flags
                     &TC); // receive thread identifier

        if( aThread[i] == NULL )
        {
            printf("CreateThread error: %d\n", GetLastError());
            return -1;
        }

            //Create event array to signal back to main that thread is ready
            //for more work
          hWorkerThreadDone[i] = CreateEvent(
                  NULL,               // default security attributes
                  TRUE,               // manual-reset event
                  FALSE,              // initial state is nonsignaled
                  NULL  // object name
                  );

            if (hWorkerThreadDone[i] == NULL)
            {
                  printf("CreateEvent failed (%d)\n", GetLastError());
                  return -1;
            }

            //Create event array to signal to worker threads that they can
            //resume work
          hEventMoreWorkToDo[i] = CreateEvent(
                  NULL,               // default security attributes
                  TRUE,               // manual-reset event
                  FALSE,              // initial state is nonsignaled
                  NULL  // object name
                  );

            if (hEventMoreWorkToDo[i] == NULL)
            {
                  printf("CreateEvent failed (%d)\n", GetLastError());
                  return -1;
            }

    }

      t = (double)cvGetTickCount() - t;
      printf( "Thread Creation time = %gms\n",
t/((double)cvGetTickFrequency()*1000.) );

      DWORD dwRet;
```

```
	while(1)
	{
		//printf("Main thread waiting on worker threads to
finish....\n");
		//Wait for worker thread to complete

		t = (double)cvGetTickCount();
		dwRet = WaitForMultipleObjects(THREADCOUNT, hWorkerThreadDone,
TRUE, INFINITE);
		//End time for processing

		t = (double)cvGetTickCount() - t;
	  printf( "detection time = %gms\n",
t/((double)cvGetTickFrequency()*1000.) );

		//Average speed
		avgSpeed +=t;
		numFrame++;
	       printf("Average = %f\n",
(avgSpeed/(double)numFrame)/(((double)cvGetTickFrequency()*1000.)));

		if(numFrame==100)
			break;

		if( WAIT_OBJECT_0 == dwRet)
		{
			//Load frame

			if(!img) break;

			//Filter frame
			cvCvtColor(img,g_gray,CV_RGB2GRAY);

			//char c = cvWaitKey(0);

			//Reset the hWorkerThreadDone events
			for(int i=0; i < THREADCOUNT; i++ )
				ResetEvent(hWorkerThreadDone[i]);

			//Signal worker thread to process next frame
			for(int i=0; i < THREADCOUNT; i++ )
				SetEvent(hEventMoreWorkToDo[i]);
		}
	}

	cvSaveImage("out4T.jpg",g_output );
	cvThreshold(g_output,g_gray,75,255,CV_THRESH_BINARY);
	cvSaveImage("out4T_thres.jpg",g_gray );

	//Clean up
	char c;
	cin >> c;
	cvWaitKey(0);

	cvReleaseImage(&img);
	cvReleaseImage(&g_gray);
	cvReleaseImage(&g_output);
```

```
        cvReleaseImageHeader(&sub_imgTopRight_Src);
        cvReleaseImageHeader(&sub_imgTopRight_Src);
        cvReleaseImageHeader(&sub_imgBottomLeft_Src);
        cvReleaseImageHeader(&sub_imgBottomRight_Src);
        cvReleaseImageHeader(&sub_imgTopRight_Dst);
        cvReleaseImageHeader(&sub_imgTopRight_Dst);
        cvReleaseImageHeader(&sub_imgBottomLeft_Dst);
        cvReleaseImageHeader(&sub_imgBottomRight_Dst);

        cvDestroyWindow("Example1");

        return 0;

}
```

B.3. Beast 8 Threaded Code

```
/*
	main8Threaded.cpp
	NA Vandal
	April 2009
	USNA EE Final Design PJ

*/

#include "highgui.h"
#include "cv.h"
#include <iostream>
#include <process.h>
#include <windows.h>
#include "cxmisc.h"

#define IMAGE_DATA_PTR(origIm, x, y) ((origIm)->imageData + (y)*(origIm)-
>widthStep + (x)*(origIm)->nChannels)

using namespace std;

CvMat* kernel;

const int g_thres_h = 48;
const int g_thres_l = 32;
const int k=3;
const int THREADCOUNT = 8;

IplImage* g_output = NULL;
//IplImage* g_output_whole = NULL;
IplImage* g_gray = NULL;

//Sub image src/dst headers
IplImage* sub_img_src[THREADCOUNT];
IplImage* sub_img_dst[THREADCOUNT];

//Thread handles
HANDLE aThread[THREADCOUNT];
HANDLE hWorkerThreadDone[THREADCOUNT];
HANDLE hEventMoreWorkToDo[THREADCOUNT];


#define ALIGN128(i) ((i+127)&~127) //16 bytes

static CvMat *create_mat( int rows, int cols, int type )
{
	CvMat *ret;
	int is_align = 1;

	if ( is_align ) {
		ret = (CvMat*)cvAlloc( sizeof(CvMat) );
		int step = ALIGN128(cols*CV_ELEM_SIZE(type));
		void *data;
		data = cvAlignPtr(malloc( step * rows + 128 ),128);
		cvInitMatHeader( ret, rows, cols, type, data, step );
```

```
	} else {
		ret = cvCreateMat( rows, cols, type );
	}

	/*  touch page */
	memset( ret->data.ptr, 0, ret->step * ret->rows );

	return ret;
}

//Kernel constants

//Type O
const float valsOrg[] =		{	1.0000, 1.0000, 1.0000,
					1.0000, -8.0000, 1.0000,
					1.0000, 1.0000, 1.0000};

//Type H
const float sobel3by3_H[] =	{	1, 2, 1,
					0, 0, 0,
					-1, -2, -1};

//Type V
const float sobel3by3_V[] =	{	1,0,-1,
					2,0,-2,
					1,0,-1};

//Type P
const float laplacian3by3[] = {	0.1667, 0.6667, 0.1667,
					0.6667, -3.3333, 0.6667,
					0.1667, 0.667, 0.1667};
//Type L
const float log3by3_025[] =	{	3.5175, 3.5925, 3.5175,
					3.5925, -28.4397, 3.5925,
					3.5175, 3.5925, 3.5175};

const float log3by3_050[] =	{	0.4038, 0.8021, 0.4038,
					0.8021, -4.8233, 0.8021,
					0.4038, 0.8021, 0.4038};

const float log3by3_075[] =	{	0.2189, 0.0292, 0.2189,
					0.0292, -0.9926, 0.0292,
					0.2189, 0.0292, 0.2189};


const float log5by5_025[] =	{	1.2663, 1.2663, 1.2663, 1.2663, 1.2663,
					1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
					1.2663, 1.3413, -30.6908, 1.3413, 1.2663,
					1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
					1.2663, 1.2663, 1.2663, 1.2663, 1.2663};

const float log5by5_050[] =	{	0.0448, 0.0468, 0.0564, 0.0468, 0.0448,
					0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
					0.0564, 0.7146, -4.9048, 0.7146, 0.0564,
					0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
					0.0448, 0.0468, 0.0564, 0.0468, 0.0448};
```

```
const float log5by5_075[] =   {     0.0059, 0.0416, 0.0743, 0.0416, 0.0059,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0743, -0.0451, -1.0058, -0.0451,0.0743,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0059, 0.0416, 0.0743, 0.0416, 0.0059};


const float log7by7_025[] =   {     0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.7211  -31.3110, 0.7211, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461};

const float log7by7_050[] =   {     0.0228, 0.0228, 0.0228, 0.0229, 0.0228,
0.0228, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0229, 0.0345, 0.6927   -4.9267, 0.6927, 0.0345, 0.0229,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0228, 0.0228, 0.0229, 0.0228, 0.0228, 0.0228};

const float log7by7_075[] =   {     0.0001, 0.0002, 0.0011, 0.0024, 0.0011,
0.0002, 0.0001,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0001, 0.0002, 0.0011, 0.0024, 0.0011, 0.0002, 0.0001};

const float log9by9_025[] =   {     0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.4659  -31.5663, 0.4659, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908};

const float log9by9_050[] =   {     0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0254, 0.6837   -4.9357, 0.6837, 0.0254, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138};
```

```
const float log9by9_075[] =   {     0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};

static CvMat* getAlignedKernel(char type, int size=3, int coeff = 2)
{
      // *((float *)(kernel->data.ptr + i * kernel->step + j))=vals[c++];

      if(size!=3 && size!=5 && size!=7 && size!=9)
      {
            cout<<"Error: Invalid Kernel Size!\n";
            exit(-1);
      }

      CvMat* kernel;

      if(type=='O')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)
                        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=valsOrg[c++];
      }
      else if(type=='H')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)

      *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_H[c++];

      }
      else if(type=='V')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)

      *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_V[c++];

      }
      else if(type=='P')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=laplacian3by3[c++];

}
else if(type=='L')
{
	kernel= create_mat(size,size,CV_32FC1);

	if(size==3)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_050[c++];
		}
		else if(coeff==3)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_075[c++];
		}
		else
		{
			cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
			exit(-1);
		}
	}
	else if(size==5)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_050[c++];
```

```
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==7)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }

      }
      else if(size==9)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_025[c++];
```

```
                }
                else if(coeff==2)
                {
                        int c=0;
                        for(int i=0; i<size; ++i)
                                for(int j=0; j<size; ++j)

        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_050[c++];
                }
                else if(coeff==3)
                {
                        int c=0;
                        for(int i=0; i<size; ++i)
                                for(int j=0; j<size; ++j)

        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_075[c++];
                }
                else
                {
                        cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                        exit(-1);
                }
        }
        else
        {
                cout<<"Error: Invalid Kernel Size!\n";
                exit(-1);
        }
    }
    else
    {
        cout<<"Error: Invalid Kernel Type!\n";
        exit(-1);
    }



    return kernel;
}


unsigned __stdcall helperThreadFunc( void* pArguments )
{
    int q = (int)pArguments;
    DWORD dwRet;

    //printf("In worker thread %i...\n",q);
    do{

        //Do  work on specific quadrant based upon argument

        cvFilter2D(sub_img_src[q],sub_img_dst[q],kernel);

        //Signal that has completed work, wait for moreWork event from
        //main thread
        if (! SetEvent(hWorkerThreadDone[q]))
```

```
            {
                  //printf("SetEvent failed (%d)\n", GetLastError());

                  _endthreadex( -1 );
                  return -1;
            }

            //printf("Worker thread %i waiting for signal to resume
            //processing...\n",q);
            dwRet = WaitForSingleObject(hEventMoreWorkToDo[q],INFINITE);

      }while(dwRet == WAIT_OBJECT_0);

      _endthreadex( 0 );

      return 0;

}

int main(int argc, char ** argv)
{
      double avgSpeed = 0;
      long numFrame = 0;

      const char* filename = argc >=2 ? argv[1] : "USMC.jpg";
      double t;

      //Create kernel
      kernel = getAlignedKernel('L',9,2);

      IplImage* img = cvLoadImage(filename);

      g_gray = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
      g_output = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);

      cvCvtColor(img,g_gray,CV_BGR2GRAY);

      //***Setup sub regions --- Source & Destination
      /*
      +---+---+---+---+
      | 0 | 1 | 2 | 3 |
      +---+---+---+---+
      | 4 | 5 | 6 | 7 |
      +---+---+---+---+
      */
      cvCvtColor(img,g_gray,CV_RGB2GRAY);

      //Time thread creation
      t = (double)cvGetTickCount();


      for(int i=0; i<THREADCOUNT; ++i)
      {
            //Initialize source and desination
            sub_img_src[i] = cvCreateImageHeader(cvSize(g_gray-
>width/4,g_gray->height/2), g_gray->depth,g_gray->nChannels );
```

```
            sub_img_dst[i] = cvCreateImageHeader(cvSize(g_output-
>width/4,g_output->height/2), g_output->depth, g_output->nChannels );

            //Origin, widthstep, imageDataOrigin
            sub_img_src[i]->origin = g_gray->origin;
            sub_img_src[i]->widthStep = g_gray->widthStep;
            sub_img_src[i]->imageDataOrigin = g_gray->imageDataOrigin;

            sub_img_dst[i]->origin = g_output->origin;
            sub_img_dst[i]->widthStep =g_output->widthStep;
            sub_img_dst[i]->imageDataOrigin = g_output->imageDataOrigin;

            //Image data
            sub_img_src[i]->imageData = IMAGE_DATA_PTR(g_gray, (g_gray-
>width/4)*(i % 4), (g_gray->height/2)*(i / 4));
            sub_img_dst[i]->imageData = IMAGE_DATA_PTR(g_output, (g_output-
>width/4)*(i % 4), (g_output->height/2)*(i / 4));

      }

      unsigned TC;
      //Load first frame
      if(!img) return -1;

      //Filter frame

      // Create worker threads
      for( int i=0; i < THREADCOUNT; i++ )
      {
            //Create worker thread
            aThread[i] = (HANDLE)_beginthreadex(
                  NULL,       // default security attributes
                  0,          // default stack size
                  &helperThreadFunc,
                  (void*)i,       //thread function arguments
                  0,          // default creation flags
                  &TC); // receive thread identifier

            if( aThread[i] == NULL )
            {
                  printf("CreateThread error: %d\n", GetLastError());
                  return -1;
            }

            //Create event array to signal back to main that thread is ready
            //for more work
            hWorkerThreadDone[i] = CreateEvent(
                  NULL,               // default security attributes
                  TRUE,               // manual-reset event
                  FALSE,              // initial state is nonsignaled
                  NULL  // object name
                  );

            if (hWorkerThreadDone[i] == NULL)
            {
                  printf("CreateEvent failed (%d)\n", GetLastError());
                  return -1;
```

```
	}

	//Create event array to signal to worker threads that they can
	//resume work
	hEventMoreWorkToDo[i] = CreateEvent(
		NULL,               // default security attributes
		TRUE,               // manual-reset event
		FALSE,              // initial state is nonsignaled
		NULL  // object name
		);

	if (hEventMoreWorkToDo[i] == NULL)
	{
		printf("CreateEvent failed (%d)\n", GetLastError());
		return -1;
	}

}

t = (double)cvGetTickCount() - t;

printf( "Thread Creation time = %gms\n",
t/((double)cvGetTickFrequency()*1000.) );

DWORD dwRet;

while(1)
{

	t = (double)cvGetTickCount();

	//Wait for worker thread to complete
	dwRet = WaitForMultipleObjects(THREADCOUNT, hWorkerThreadDone,
	TRUE, INFINITE);

	t = (double)cvGetTickCount() - t;

	printf( "Detection time = %gms\n",
	t/((double)cvGetTickFrequency()*1000.) );

	//Average speed
	avgSpeed +=t;
	numFrame++;
	printf("Average = %f\n",
	(avgSpeed/(double)numFrame)/(((double)cvGetTickFrequency()*1000.)
	));

	if(numFrame==100)
	{
		char c;
		cin >> c;

		cvSaveImage("out8T.jpg",g_output );
		cvThreshold(g_output,g_gray,75,255,CV_THRESH_BINARY);
		cvSaveImage("out8T_thres.jpg",g_gray );

		cvReleaseImage(&img);
```

```
                cvReleaseImage(&g_gray);
                cvReleaseImage(&g_output);

                for(int i=0; i<THREADCOUNT; ++i)
                {
                        cvReleaseImageHeader(&sub_img_src[i]);
                        cvReleaseImageHeader(&sub_img_dst[i]);
                }
                //Clean up

                return 0;
        };

        if( WAIT_OBJECT_0 == dwRet)
        {

                //Load frame

                if(!img) break;

                //Filter frame
                cvCvtColor(img,g_gray,CV_RGB2GRAY);


                //Reset the hWorkerThreadDone events
                for(int i=0; i < THREADCOUNT; i++ )
                        ResetEvent(hWorkerThreadDone[i]);

                //Signal worker thread to process next frame
                for(int i=0; i < THREADCOUNT; i++ )
                        SetEvent(hEventMoreWorkToDo[i]);
        }
    }
}
```

B.4. PlayStation3 Code

```
/*
      mainPS3.cpp
      NA Vandal
      April 2009
      USNA EE Final Design PJ

*/
#include "highgui.h"
#include "cv.h"
#include "cxmisc.h"

#include <iostream>
using namespace std;

#define ALIGN128(i) ((i+127)&~127) //16 bytes
//Kernel constants

//Type O
const float valsOrg[] =       {     1.0000, 1.0000, 1.0000,
                                    1.0000, -8.0000, 1.0000,
                                    1.0000, 1.0000, 1.0000};

//Type H
const float sobel3by3_H[] =   {     1, 2, 1,
                                    0, 0, 0,
                                    -1, -2, -1};

//Type V
const float sobel3by3_V[] =   {     1,0,-1,
                                    2,0,-2,
                                    1,0,-1};

//Type P
const float laplacian3by3[] = {     0.1667, 0.6667, 0.1667,
                                    0.6667, -3.3333, 0.6667,
                                    0.1667, 0.667, 0.1667};
//Type L
const float log3by3_025[] =   {     3.5175, 3.5925, 3.5175,
                                    3.5925, -28.4397, 3.5925,
                                    3.5175, 3.5925, 3.5175};

const float log3by3_050[] =   {     0.4038, 0.8021, 0.4038,
                                    0.8021, -4.8233, 0.8021,
                                    0.4038, 0.8021, 0.4038};

const float log3by3_075[] =   {     0.2189, 0.0292, 0.2189,
                                    0.0292, -0.9926, 0.0292,
                                    0.2189, 0.0292, 0.2189};


const float log5by5_025[] =   {     1.2663, 1.2663, 1.2663, 1.2663, 1.2663,
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.3413, -30.6908, 1.3413, 1.2663,
                                    1.2663, 1.2663, 1.3413, 1.2663, 1.2663,
                                    1.2663, 1.2663, 1.2663, 1.2663, 1.2663};
```

```
const float log5by5_050[] =   {     0.0448, 0.0468, 0.0564, 0.0468, 0.0448,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0564, 0.7146, -4.9048, 0.7146, 0.0564,
                                    0.0468, 0.3167, 0.7146, 0.3167, 0.0468,
                                    0.0448, 0.0468, 0.0564, 0.0468, 0.0448};

const float log5by5_075[] =   {     0.0059, 0.0416, 0.0743, 0.0416, 0.0059,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0743, -0.0451, -1.0058, -0.0451,0.0743,
                                    0.0416, 0.1332, -0.0451, 0.1332, 0.0416,
                                    0.0059, 0.0416, 0.0743, 0.0416, 0.0059};


const float log7by7_025[] =   {     0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.7211  -31.3110, 0.7211, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.7211, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461,
0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461, 0.6461};

const float log7by7_050[] =   {     0.0228, 0.0228, 0.0228, 0.0229, 0.0228,
0.0228, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0229, 0.0345, 0.6927   -4.9267, 0.6927, 0.0345, 0.0229,
0.0228, 0.0249, 0.2948, 0.6927, 0.2948, 0.0249, 0.0228,
0.0228, 0.0229, 0.0249, 0.0345, 0.0249, 0.0229, 0.0228,
0.0228, 0.0228, 0.0228, 0.0229, 0.0228, 0.0228, 0.0228};

const float log7by7_075[] =   {     0.0001, 0.0002, 0.0011, 0.0024, 0.0011,
0.0002, 0.0001,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024,
0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011,
0.0002, 0.0051, 0.0407, 0.0735, 0.0407, 0.0051, 0.0002,
0.0001, 0.0002, 0.0011, 0.0024, 0.0011, 0.0002, 0.0001};

const float log9by9_025[] =   {     0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.4659  -31.5663, 0.4659, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3909, 0.4659, 0.3909, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908,
0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908, 0.3908};

const float log9by9_050[] =   {     0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
```

```
0.0138, 0.0138, 0.0254, 0.6837   -4.9357, 0.6837, 0.0254, 0.0138, 0.0138,
0.0138, 0.0138, 0.0158, 0.2858, 0.6837, 0.2858, 0.0158, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0158, 0.0254, 0.0158, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138,
0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138, 0.0138};

const float log9by9_075[] =   {     0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0024, 0.0735   -0.0459   -1.0059   -0.0459, 0.0735, 0.0024, 0.0000,
0.0000, 0.0011, 0.0407, 0.1323   -0.0459, 0.1323, 0.0407, 0.0011, 0.0000,
0.0000, 0.0001, 0.0050, 0.0407, 0.0735, 0.0407, 0.0050, 0.0001, 0.0000,
0.0000, 0.0000, 0.0001, 0.0011, 0.0024, 0.0011, 0.0001, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000};


static CvMat *create_mat( int rows, int cols, int type )
{
    CvMat *ret;

        ret = (CvMat*)cvAlloc( sizeof(CvMat) );
        int step = ALIGN128(cols*CV_ELEM_SIZE(type));
        void *data;
        data = cvAlignPtr(malloc( step * rows + 128 ),128);
        cvInitMatHeader( ret, rows, cols, type, data, step );


    /*  touch page */
    memset( ret->data.ptr, 0, ret->step * ret->rows );

    return ret;
}


static CvMat* getAlignedKernel(char type, int size=3, int coeff = 2)
{
      // *((float *)(kernel->data.ptr + i * kernel->step + j))=vals[c++];

      if(size!=3 && size!=5 && size!=7 && size!=9)
      {
            cout<<"Error: Invalid Kernel Size!\n";
            exit(-1);
      }

      CvMat* kernel;

      if(type=='O')
      {
            kernel= create_mat(3,3,CV_32FC1);
            int c=0;
            for(int i=0; i<3; ++i)
                  for(int j=0; j<3; ++j)
                        *((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=valsOrg[c++];
      }
      else if(type=='H')
```

```
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_H[c++];

}
else if(type=='V')
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=sobel3by3_V[c++];

}
else if(type=='P')
{
	kernel= create_mat(3,3,CV_32FC1);
	int c=0;
	for(int i=0; i<3; ++i)
		for(int j=0; j<3; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=laplacian3by3[c++];

}
else if(type=='L')
{
	kernel= create_mat(size,size,CV_32FC1);

	if(size==3)
	{
		if(coeff==1)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_025[c++];
		}
		else if(coeff==2)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_050[c++];
		}
		else if(coeff==3)
		{
			int c=0;
			for(int i=0; i<size; ++i)
				for(int j=0; j<size; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log3by3_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==5)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log5by5_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else if(size==7)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)
```

```
*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log7by7_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }

      }
      else if(size==9)
      {
            if(coeff==1)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_025[c++];
            }
            else if(coeff==2)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_050[c++];
            }
            else if(coeff==3)
            {
                  int c=0;
                  for(int i=0; i<size; ++i)
                        for(int j=0; j<size; ++j)

*((float*)CV_MAT_ELEM_PTR(*kernel,i,j))=log9by9_075[c++];
            }
            else
            {
                  cout<<"Error: Invalid Coeff {1,2 or 3}!\n";
                  exit(-1);
            }
      }
      else
      {
            cout<<"Error: Invalid Kernel Size!\n";
            exit(-1);
      }
}
else
```

```
	{
		cout<<"Error: Invalid Kernel Type!\n";
		exit(-1);
	}

	return kernel;
}

int main(int argc, char ** argv)
{
	double avgSpeed = 0;
	long numFrame = 0;

  const char* filename = argc >=2 ? argv[1] : "USMC.jpg";
   double t;

   CvMat * kernel = getAlignedKernel('L',3,2);

	IplImage* img = cvLoadImage(filename);

	IplImage* gray = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
	IplImage* out = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);

		cvCvtColor(img,gray,CV_BGR2GRAY);

		while(1)
		  {

		    t = (double)cvGetTickCount();

		    cvFilter2D(gray,out,kernel);

		    t = (double)cvGetTickCount() - t;

		    //Average speed
		    avgSpeed +=t;
		    numFrame++;

		    if(numFrame==100)
		      {

			printf("Average = %f\n",
			(avgSpeed/(double)numFrame)/(((double)cvGetTickFrequ(
			)
			*1000.)));

			break;
		      }

		  }
		cvSaveImage("outPS3.jpg",out );
		cvThreshold(out,out,25,255,CV_THRESH_BINARY);
		cvSaveImage("outPS3_thres.jpg",out );

	cvReleaseImage(&img);
	cvReleaseImage(&gray);
```

```
	cvReleaseImage(&out);
	return 0;
}
```

B.5. PlayStation3 Makefile

```
main:
	g++ main.cpp `pkg-config --cflags --libs opencv` -o run
```

B.6. PlayStation3 test.sh

```
#!/bin/bash

rm *_outPS3*.jpg
clear

echo Running OpenCV on the Cell Filter2D Test....
echo
echo

echo 0 SPEs
echo -----------------------
echo tux.jpg
CVCELL_SPENUM=0 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=0 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=0 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=0 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=0 ./run road.jpg
echo

echo 1 SPEs
echo -----------------------
echo tux.jpg
CVCELL_SPENUM=1 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=1 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=1 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=1 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=1 ./run road.jpg
echo

echo 2 SPEs
echo -----------------------
echo tux.jpg
CVCELL_SPENUM=2 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=2 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=2 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=2 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=2 ./run road.jpg
echo

echo 3 SPEs
echo -----------------------
echo tux.jpg
```

```
CVCELL_SPENUM=3 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=3 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=3 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=3 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=3 ./run road.jpg
echo

echo 4 SPEs
echo ----------------------
echo tux.jpg
CVCELL_SPENUM=4 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=4 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=4 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=4 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=4 ./run road.jpg
echo

echo 5 SPEs
echo ----------------------
echo tux.jpg
CVCELL_SPENUM=5 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=5 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=5 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=5 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=5 ./run road.jpg
echo

echo 6 SPEs
echo ----------------------
echo tux.jpg
CVCELL_SPENUM=6 ./run tux.jpg
echo AppleII.jpg
CVCELL_SPENUM=6 ./run AppleII.jpg
echo electricity.jpg
CVCELL_SPENUM=6 ./run electricity.jpg
echo USMC.jpg
CVCELL_SPENUM=6 ./run USMC.jpg
echo road.jpg
CVCELL_SPENUM=6 ./run road.jpg
echo

echo
echo Testing complete!
echo
```

```
ls -1 *_outPS3*.jpg

echo
echo Use "xv <filename>" to view ...
echo
echo
```