# PARALLELIZING AwSpPCA FOR ROBUST FACIAL RECOGNITION USING CUDA

By

Syed Amer Zawad

and

Ashfaque Ali

A Thesis Submitted in Partial Fulfillment of the Requirements for the Bachelors in Science Degree

Department of Computer Science and Engineering

BRAC University, April 2014

## THESIS APPROVAL

PARALLELIZING AW SP PCA FOR ROBUST FACIAL RECOGNITION USING CUDA

By Syed Amer Zawad and Ashfaque Ali

This is to certify that thesis is a presentation of our original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions.

This Thesis is Submitted in Partial Fulfillment of the Requirements for the Bachelors of Science Degree in Computer Science and Engineering

Approved By

Md. Zahangir Alom (Supervisor) ________________________

Professor Md. Zahidur Rahman (Chairperson) ________________________

Date of Approval

____________________________________

Thesis submitted by

Syed Amer Zawad (Student ID: 13301094)

Ashfaque Ali (Student ID: 10101009)

For the B.Sc in Computer Science and Engineering Degree, presented on 30th April 2014, in BRAC University, Dhaka, Bangladesh

Supervisor – Md. Zahangir Alom

Co-supervisor – Risul Karim

# TABLE OF CONTENTS

# Abstract

This paper was conducted to analyze the performance benefits of parallelizing the Adaptive Weighted Sub-patterned Principle Component Analysis (Aw SP PCA) algorithm, given that the algorithm is implemented so as to retain the accuracy from its serialized version. The serialized execution of this algorithm is analyzed first and then compared against its parallel implementation, both compiled and run on the same computer. Throughout this paper, the methodology is to undergo a step by step procedure which can clearly outline and describe the problems faced when trying to parallelize this algorithm. It will also describe where, how and why parallelizing procedures were used.

The results of the research have shown that while not all parts of the algorithm can be implemented in parallel in the first place, some of the sections that can be parallelized does not necessarily yield a considerable amount of benefits. Also, it was seen that not all sections scale well with problem size, meaning that some portions of the algorithm can be left in its serialized state without much loss in time. The sections which can be parallelized were discussed in detail. Some changes were also made to certain variables to ensure the best accuracy possible. Finally, through analysis and experimentation, a speedup of 2.76 was achieved, with a recognition accuracy of 92.6%.

# Acknowledgements

We would like to thank Md. Zahangir Alom for agreeing to supervise us with our thesis. His patience and confidence in us has been a source of encouragement and without his guidance this paper would not have come into existence. We believe his dedication to this paper deserves to be reciprocated with great gratitude.

We would like to thank Risul Karim for supporting us as co-supervisor in our thesis and Md. Shamsul Kaonine for allocating university resources for our use for this paper.

A special thanks to the Thesis committee for taking the time to review and evaluate our thesis as part of our undergraduate program.

# List of Figures and Tables

# 1. Introduction

## 1.1 Motivation

The reason behind undertaking this research is to better understand and evaluate the hardware capabilities of modern day commercially produced Graphics Processing Units. A few years ago, parallel implementations of programs required the use of multiple computers working over a network with each other in order to emulate a parallel programming environment. This needed the use of networking as well as good use of every individual processing unit. Nowadays, this kind of power is readily available to us, in the form of multiprocessor CPUs and GPUs. While the network models for parallel processing are still widely in use, one does not need the amount of physical space necessary for the networked models to be able to enjoy the benefits of parallel processing. This is mainly due to modern day advancements in the hardware industry and the availability of parallel processing capabilities in mass-produced processors. This is a major leap in the field of computer science as it allows for faster solutions to classical problems. However, it is understood that not all programs can be implemented this way and not all problems have parallel solutions. Thus every algorithm needs to be evaluated for its “parallelizability”.

Aw SP PCA is an update on the traditional PCA algorithm and used in facial recognition. The main motivation for implementing this algorithm using parallel programming techniques is because of its significance in the history of Facial Recognition and also because of the opportunities it provides us. At first glance, it seems very obvious that certain parts of this algorithm are so promising that it might be considered “embarrassingly parallel”. We believe that this algorithm is a good example of how a traditional algorithm might use the powers of modern day hardware to perform better. An evaluation of the parallelization process might also provide us an insight into the world of parallel programming, which (as undergraduate students) we see as a great opportunity to further ourselves in the field of computer science.

## 1.2 Related Works

There are many face recognition proposals employing PCA [9]. P. Shamna et al. and W. Zhao et al. indicated that most of them used PCA for different purposes and obtained several distinctive features, e.g., less memory requirement and simple computation complexity [5-7, 15, 19-21]. The results of the survey brought about various PCA derivations including increasing the recognition rate. For example, in 2010, E. Gumus et al. [22] applied a hybrid approach over PCA and Wavelets to extract feature resulting in higher recognition rate. Recently, some research have also improved the recognition rate; for instance, A. K. Bansal and P. Chawla [23], in 2013, have proposed Normalized Principal Component Analysis (NPCA) which normalized images to remove the lightening variations and background effects by applying SVD instead of eigenvalue decomposition.

Furthermore, in the same year, X. Yue [24] proposed to use a radial basis function to constructa kernel matrix by computing the distance of two different vectors calculated by the parameter of 2-norm exponential, and then apply a cosine distance to calculate the matching distance leading to higher recognition rate over a traditional PCA. Similarly, L. Min et al. [16] introduced a two dimensional concept for PCA (2DPCA) for face feature extraction to maintain the recognition rate but with lower computational recognition time. Note that only a few proposals investigated on a computational time complexity.

Recently in 2013, G. D. C. Cavalcanti et al. [33] proposed a novel method called (weighted) Modular IMage PCA by dividing a single image into different modules to individually recognize human face to reduce computational complexity. Previously, in 2003, J. Chunghong et al. [34] proposed a distributed parallel system for face recognition by dividing trained face databases into five sub-databases feeding into each individual computer hardware system, and then performed an individual face recognition algorithm individually in parallel over TCP/IP socket communication, and after that, the recognition feedbacks are sent back for making a final decision at the master. Similarly, X. Liu and G. Su [35] modified a distributed system to support parallel retrieval virtual machines by allowing multi-virtual machines for each slave to run individual face recognition algorithms.

## 1.3 Methodology

The paper starts with the question "How much does the Aw SP PCA performance improve if implemented in parallel?" and hopes to answer this through quantitative analysis of the research results. We also intend to find out how much of an impact modern hardware can make on this algorithm. As such, we follow a very methodical analysis of every portion of the algorithm and try to answer the proposed questions based completely on our evaluations. We begin by implementing the basic Aw Sp Principle Component Analysis in its traditional serialized form. To understand the parallelization process better, we further segment the code into sections (see chapter 3). These sections are evaluated individually through a time graph analysis. Aw SP PCA was implemented to calculate the Eigen space. Then we implement the recognition section which applies the face-space projection and Euclidean distance measurements to find the database image which is the test image's closest match. This same program is run again on the same machine, except this time we used functions from MATLAB's Parallel Processing Toolbox (see chapter 2.6) to execute the program in parallel using CUDA on the NVIDIA GPU. The graphs are also plotted for this and the results are compared between the two implementations for every section. The results are discussed in Chapter 4.

## 1.4 Outline

Chapter 2 provides all the relevant information one may require in order to understand the terminologies, techniques and objects used for this paper. The literature review was written so as to assist the reader in case they require it regarding the information in this paper. It explains the algorithms involved in detail, as well as defining the parallel processing techniques used. Finally, it contains information regarding the libraries used and the programming environment which was used to run the code.

Chapter 3 describes the implementation details that were performed when running the code. It discusses the algorithm and how it functions in a step-by-step manner. The whole process of the implementation is explained thoroughly and snippets of the code are shown where appropriate.

Chapter 4 discusses the results obtained from the previous chapter. It first elaborates on the database used then talks about the runtime environment in details. It explains the output of the Aw SP PCA code and evaluates them in context of parallel programming.

Chapter 5 draws a conclusion from the evaluations of the previous chapter and tries to settle the questions posed by the research paper. It also extends into providing suggestions as to what can be done to further improve the results of this paper and proposes a few more questions.

Chapter 6 lists all the related works used in the production of this paper and any and all citations provided.

# 2. Literature Review

## 2.1 Aw SP PCA

Principal component analysis (PCA) is an algorithm that has found application in fields such as face recognition and image compression, and is a common technique for finding patterns in data of high dimension. The luxury of graphical representation is not available when searching for patterns in data of high dimensions. This is where PCA becomes a powerful tool for analyzing data.

The traditional PCA operates directly on whole patterns represented as (feature) vectors to extract so-needed global features for subsequent classification by a set of previously found global projectors from a given training pattern set, whose aim is to maximally preserve original pattern information after extracting features, i.e., reducing dimensionality. SpPCA operates instead directly on a set of partitioned sub-patterns of the original pattern and acquires a set of projection sub-vectors for each partition to extract corresponding local sub-features and then synthesizes them into global features for subsequent classification. These sub-patterns are formed via a partition for an original whole pattern and utilized to compose multiple training sub-pattern sets for the original training pattern set. In this way, SpPCA can independently be performed on individual training sub-pattern sets and finds corresponding local projection sub-vectors, and then uses them to extract local sub-features from any given pattern. Afterwards, these extracted sub-features from individual sub-patterns are synthesized into a global feature of the original whole pattern for subsequent classification.

Aw-SpPCA operates directly on the sub patterns partitioned from an original whole pattern and separately extracts features from them. In Aw-SpPCA not only is the spatially-related information in a face image considered and preserved in each sub-pattern, but also the different contributions made by different parts of the face are emphasized. Aw-SpPCA incorporates both PCA and SPPCA algorithms. Where SpPCA does not concern different contributions made by different sub-patterns, in other words, it endows equal importance to different parts of a pattern in classification, Aw-SpPCA can adaptively compute the contributions of each part and then endows them to a classification task in order to enhance the robustness to both expression and illumination variations. Due to this classification accuracies are improved beyond that of SpPCA.

## 2.2 Parallel computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously.[23]It defines the ability to complete multiple tasks concurrently. Up till now parallelism has only been used to some extent. However, in the age of high performance it is absolutely necessary for programs to run faster and when everything else remains constant parallel computing is the only way to achieve better performance. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling.[24]Frequency scaling was the dominant reason for improvements in computer performance. The runtime of a program is equal to the number of instructions multiplied by the average time per instruction. Maintaining everything else constant, increasing the clock frequency decreases the average time it takes to execute an instruction. However increases in frequency increase the amount of power used in a processor. To solve both the power consumption and frequency scaling problems, parallel computing is necessary. It operates on the principle that large problems can often be divided into smaller ones, which can be solved with relative ease. These small problems then are distributed among the processing units to be solved concurrently with the objective of reducing the runtime of the program. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.[25]Ideally, parallel processing makes programs run faster because there are more engines (CPUs or Cores) running it. In practice, it is often difficult to divide a program in such a way that separate CPUs or cores can execute different portions without interfering with each other. However when program environment stays constant the only way to improve a program's execution speed is through parallel computing.

Theoretically, parallelization induced speed-up should be linear as in, doubling the number of processing elements should halve the runtime, and doubling it a second time should halve the current runtime. However, in practice very few parallel algorithms can achieve this. Most of them have a near-linear speed-up for small numbers of processing elements, which flattens out into a constant value for large numbers of processing elements. The potential speed-up of an algorithm on a parallel computing platform is given by Amdahl's law. It states that a small portion of the program which cannot be parallelized will limit the overall speed-up available from parallelization. A program solving a large mathematical or engineering problem will typically consist of several parallelizable parts and several non-parallelizable (sequential) parts. If the sequential portion of a program accounts for 10% of the runtime, we can get no more than a 10× speed-up, regardless of how many processors are added. This puts an upper limit on the usefulness of adding more parallel execution units.

When a task cannot be partitioned because of sequential constraints, the application of more effort has no effect on the schedule. The bearing of a child takes nine months, no matter how many women are assigned.[26]

One of the fundamental concepts, anyone trying to implement parallel algorithms has to understand is data dependencies. No program can run more quickly than the longest chain of dependent calculations, since calculations that depend upon prior calculations in the chain must be executed in order. However, most algorithms do not consist of just a long chain of dependent calculations; there are usually opportunities to execute independent calculations in parallel. So if there are no dependencies between instructions or tasks, then they can be executed in parallel.

There are several different forms of parallel computing: bit-level, instruction-level parallelism and task parallelism. Ever since very-large-scale integration computer-chip fabrication technology had been invented, speed-up in computer architecture was done by increasing bit-level parallelism. Increasing the word or bit size reduces the number of instructions the processor must execute to perform an operation on variables whose sizes are greater than the length of the word. For example, an 8-bit processor takes two instructions to complete a 16-bit integer addition, where a 16-bit processor can complete the operation with a single instruction. In instruction-level parallelism(ILP), the number of operations in a program that can be run simultaneously is measured. The goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Where typically programs are written in a sequential manner and executes one after the other, ILP allows compiler and the processor to overlap the execution of multiple instructions or change the order in which they are to be executed.

Task parallelism focuses on distributing execution process on a program code across different parallel computing units in parallel computing environments. In a multiprocessor system, task parallelism is achieved when each processor executes multiple threads containing same or different code, on the same or different data. In general, different execution threads communicate with one another as they work. Communication usually takes place by passing data from one thread to the next as part of a work-flow. Task parallelism emphasizes on the distribution of the processes, as opposed to the data parallelism which focuses on distributing data. Most real programs fall somewhere on a continuum between task parallelism and data parallelism.

Parallel computers can be classified according to the level at which the hardware supports parallelism. Among these classifications General-purpose computing on graphics processing units (GPGPU) is a fairly recent trend in computer engineering research. General-Purpose Computing on Graphics Processing Units (GPGPU) is the utilization of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU).[27]

GPUs were born for high end graphics processing. Recently interfaces have been built to interact with codes not related to graphical purposes, for example for linear algebraic matrix manipulations. Because of this high-performance computing using GPUs has become increasingly popular due to their remarkable computational power, improved programmability and relatively cheap prices. Performing general purpose computing on graphics processor units usually leads to performance gains of several orders of magnitude compared with traditional CPU implementations. The model for GPU computing is to use a CPU and GPU together in a co-processing computing model. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. From the user's perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance.

GPUs are co-processors that have been heavily optimized for computer graphics processing.[28]The success of GPGPUs in the past few years has been the ease of programming of the associated CUDA parallel programming model. In this programming model, the developer modifies their application to take the compute-intensive kernels and map them to the GPU. The rest of the application remains on the CPU. Mapping a function to the GPU involves rewriting the function to expose the parallelism in the function and adding keywords to move data to and from the GPU.

Though a program may execute faster, parallel computing can also induce bugs in the program such as, race condition, deadlocks, parallel slowdown etc. In a parallel program subtasks are referred as threads or processes. Threads will often need to use or update variables that are shared among all the threads in the thread-pool. So if one thread is executing an instruction that changes the data of a variable, while another thread also manipulates that same variable then the first thread will produce the wrong result. This is known as race condition. This is easily solved by with mutual exclusion. A lock is placed on the thread to take control of a variable and prevent other threads from reading and writing it, until that variable is unlocked. After the thread has finished executing, the data is unlocked and can be accessed by other threads again. Locking multiple variables using non-atomic locks introduces the possibility of program deadlock. An atomic lock locks multiple variables all at once. If it cannot lock all of them, it does not lock any of them. If two threads each need to lock the same two variables using non-atomic locks, it is possible that one thread will lock one of them and the second thread will lock the second variable. In such a case, neither thread can complete, and deadlock results.

Parallelization does not always result in speedup. As a task is split up into smaller tasks and assigned to threads, those threads spend an increasing amount of their time communicating with each other. Eventually, the overhead from communication dominates the time spent solving the problem and further parallelization increases rather than decreases the amount of time required to finish. This is known as parallel slowdown.

## 2.3 Facial Recognition

Comparing with other biometrics, the most superiority of face biometric is its non-intrusive nature. Therefore, face is one of the most suitable biometrics for surveillance applications. A facial recognition system is a computer application which uses a digital image or a video frame from a video source to identify or verify a person. One of the ways to do this is by comparing selected facial features from the image and a facial database.

It is typically used in security systems and can be compared to other biometrics such as fingerprint or eye iris recognition systems.[29]There are different ways a facial recognition system can work. Some facial recognition algorithms identify facial features by extracting landmarks, or features, from an image of the subject's face. These features are then used to search for other images with matching features. Other algorithms normalize a gallery of face images and then compress the face data, only saving the data in the image that is useful for face recognition. A probe image is then compared with the face data. One of the earliest successful systems is based on template matching techniques applied to a set of salient facial features, providing a sort of compressed face representation. Geometric, which looks at distinguishing features, or photometric, which is a statistical approach that distills an image into values and compares the values with templates to eliminate variances are the two main approaches facial recognition can be classified into. Popular recognition algorithms include Principal Component Analysis using eigenfaces, Linear Discriminate Analysis, Elastic Bunch Graph Matching using the Fisherface algorithm, the Hidden Markov model, the Multilinear Subspace Learning using tensor representation, and the neuronal motivated dynamic link matching.

Existing methods of facial recognition can be divided into two categories: 2D methods and 3D methods (or their hybrid). Because the pose variation is essentially caused by the 3D rigid motion of face, 3D model based methods generally have higher precision than 2D methods. Due to lacking one degree of freedom, 2D methods often use some 2D transformations to approximate the 3D transformation and compensate the error by some statistical learning strategies. The learning procedures are either conducted in image space or feature space. 3D methods are always based on a 3D face model, which may be a single model, or a deformable model in certain parametric forms. The flexibility and precision of the 3D face model is the core of 3D methods, therefore we usually call them as 3D model assisted methods. In typical face recognition applications, the enrolled face images (gallery) are usually captured under controlled environment, while the quality of on-site face images (probe) are uncontrolled.

Among the newly emerging forms of facial recognition, claimed to achieve improved accuracies, is three-dimensional face recognition. This technique uses 3D sensors to capture information about the shape of a face. This information is then used to identify distinctive features on the surface of a face, such as the contour of the eye sockets, nose, and chin. One advantage of 3D facial recognition is that it is not affected by changes in lighting like other techniques. It can also identify a face from a range of viewing angles, including a profile view. Three-dimensional data points from a face vastly improve the precision of facial recognition. 3D research is enhanced by the development of sophisticated sensors that do a better job of capturing 3D face imagery. The sensors work by projecting structured light onto the face. Up to a dozen or more of these image sensors can be placed on the same CMOS chip—each sensor captures a different part of the spectrum.[30]Even a perfect 3D matching technique could be sensitive to expressions. For that goal a group at the Technion applied tools from metric geometry to treat expressions as isometries. Using the visual details of the skin, as captured in standard digital or scanned images is another emerging technique for facial recognition. This technique, called skin texture analysis, turns the unique lines, patterns, and spots apparent in a person's skin into a mathematical space.

Tests have shown that with the addition of skin texture analysis, performance in recognizing faces can increase 20 to 25 percent.[31]Questions have been raised on the effectiveness of facial recognition software in cases of railway and airport security. Face recognition is not perfect and struggles to perform under certain conditions. From the early stages of face recognition research to now, pose variation was always considered as an important problem. The problem gained great interest in the computer vision and pattern recognition research community, and many promising methods have been proposed to tackle the problem of recognizing faces in arbitrary poses. However, none of them is free from limitations and is able to fully solve the pose problem. As noted in a recent survey, the protocols for testing face recognition across pose are even not unified, which indicates that a lot more work is needed to build a fully pose invariant face recognition system. Other conditions where face recognition does not work well include poor lighting, sunglasses, long hair, or other objects partially covering the subject's face, and low resolution images. Because of these conditions and due to the limitations of the machine if the picture that is provided and the picture that is in the database vary by a considerable margin then even if the picture belongs to the same person the machine will fail to recognize the person. Another serious disadvantage is that many systems are less effective if facial expressions vary. Even a big smile can render the system less effective. For instance: Canada now allows only neutral facial expressions in passport photos. There is also inconstancy in the datasets used by researchers. Researchers may use anywhere from several subjects to scores of subjects, and a few hundred images to thousands of images. It is important for researchers to make available the datasets they used to each other, or have at least a standard dataset.

Ralph Gross, a researcher at the Carnegie Mellon Robotics Institute, describes one obstacle related to the viewing angle of the face: "Face recognition has been getting pretty good at full frontal faces and 20 degrees off, but as soon as you go towards profile, there've been problems."[32]

Authorities have found a number of applications for facial recognition systems. While earlier post-9/11 deployments were well publicized trials, more recent deployments are rarely written about due to their covert nature. The Mexican government employed facial recognition software to prevent voter fraud. Some individuals had been registering to vote under several different names, in an attempt to place multiple votes. By comparing new facial images to those already in the voter database, authorities were able to reduce duplicate registrations. Similar technologies are being used in the United States to prevent people from obtaining fake identification cards and driver’s licenses. There are also a number of potential uses for facial recognition that are currently being developed. This technology could also be used as a security measure at ATMs. Instead of using a bank card or personal identification number, the ATM would capture an image of the customer's face, and compare it to the account holder's photo in the bank database to confirm the customer's identity. Facial recognition systems are used to unlock software on mobile devices. An independently developed Android Marketplace app called VisidonApplock makes use of the phone's built-in camera to take a picture of the user. Facial recognition is used to ensure only this person can use certain apps which they choose to secure. Face detection and facial recognition are integrated into the iPhoto application for Macintosh, to help users organize and caption their collections. Modern digital cameras often incorporate a facial detection system that allows the camera to focus and measure exposure on the face of the subject, thus guaranteeing a focused portrait of the person being photographed. Some cameras, in addition, incorporate a smile shutter, or take automatically a second picture if someone closed their eyes during exposure. Because of certain limitations of fingerprint recognition systems, facial recognition systems are used as an alternative way to confirm employee attendance at work for the claimed hours. Another use could be a portable device to assist people with prosopagnosia in recognizing their acquaintances.

At Super Bowl XXXV in January 2001, police in Tampa Bay, Florida used Viisage facial recognition software to search for potential criminals and terrorists in attendance at the event. 19 people with minor criminal records were potentially identified.[33]

One of the key advantages of facial recognition is that it does not require the cooperation of the test subject to work. Properly designed systems installed in airports, multiplexes, and other public places can identify individuals among the crowd, without passers-by even being aware of the system. Other biometrics like fingerprints, iris scans, and speech recognition cannot perform this kind of mass identification. However, privacy has become one of the many issues that have been introduced due to this. It can be used not just to identify an individual, but also to unearth other personal data associated with an individual – such as other photos featuring the individual, blog posts, social networking profiles, Internet behavior, travel patterns, etc. – all through facial features alone. Moreover, individuals have limited ability to avoid or thwart facial recognition tracking unless they hide their faces. This fundamentally changes the dynamic of day-to-day privacy by enabling any marketer, government agency, or random stranger to secretly collect the identities and associated personal information of any individual captured by the facial recognition system.

## 2.4 CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce.[34]

CUDA gives program developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA supported GPUs. Using CUDA, the GPUs can be used for general purpose processing (i.e., not exclusively graphics); this approach is known as GPGPU. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. CUDA provides both a low level API and a higher level API. CUDA works with all Nvidia GPUs from the G8x series onwards, including GeForce, Quadro and the Tesla line. CUDA is compatible with most standard operating systems. Nvidia states that programs developed for the G8x series will also work without modification on all future Nvidia video cards, due to binary compatibility.CUDA has several advantages over traditional general-purpose computation on GPUs (GPGPU) using graphics APIs. Scattered read is a term used to define the ability to read from arbitrary address in memory. This is implemented in CUDA for better performance. It also has unified virtual memory. This feature was added for CUDA 6 which allows disparate x86 and GPU memory pools to be addressed together in a single space. However unified virtual addressing only simplified memory management; it did not get rid of the required explicit memory copying and pinning operations necessary to bring over data to the GPU first before the GPU could work on it. It can also download and readback, to and from the GPU faster. Full support for integer and bitwise operations, including integer texture lookups is also integrated in CUDA. Also CUDA exposes a fast shared memory region (up to 48KB per Multi-Processor) that can be shared amongst threads. Shared memory is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. So calculations and processes run faster without hampering other processes.This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.[35]

Although CUDA has many advantages it is not immune to drawbacks. For instance texture rendering is not supported by CUDA. However CUDA 3.2 and up addresses this by introducing "surface writes" to CUDA arrays, the underlying opaque data structure. Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency. This can be partly alleviated with asynchronous memory transfers, handled by the GPU's DMA engine. Also threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands. Branches in the program code do not affect performance significantly, provided that each of 32 threads takes the same execution path. the SIMD execution model becomes a significant limitation for any inherently divergent task. Valid C/C++ may sometimes be flagged and prevent compilation due to optimization techniques the compiler is required to employ to use limited resources. Double precision floats deviate from the IEEE 754 standard: round-to-nearest-even is the only supported rounding mode for reciprocal, division, and square root. In single precision, denormals and signaling NaNs are not supported. Only two IEEE rounding modes are supported (chop and round-to-nearest even), and those are specified on a per-instruction basis rather than in a control word; and the precision of division/square root is slightly lower than single precision. CUDA also runs the host code through a C++ compiler. Due to this some valid C instructions which are invalid for C++ fail to compile. Unlike OpenCL, CUDA-enabled GPUs are only available from Nvidia.[36]

## 2.5 NVidia

Nvidia Corporation is an American global technology company based in Santa Clara, California. Nvidia manufactures graphics processing units (GPUs), as well as having a significant stake in manufacture of system-on-a-chip units (SOCs) for the mobile computing market. In addition to GPU manufacturing, Nvidia provides parallel processing capabilities to researchers and scientists that allow them to efficiently run high-performance applications.GeForce is a brand of graphics processing units (GPUs) designed by Nvidia. The 700 series is a member of the GeForce family. The GeForce 700 Series contains features from both GK104 and GK110. GK110 has been designed and is being marketed with compute performance in mind. It contains 7.1 billion transistors. This model also attempts to maximise energy efficiency through the performance of as many tasks as possible in parallel according to the capabilities of its streaming processors. With GK110, Nvidia also reworked the GPU texture cache to be used for compute. There are also new features from GK110 such as CUDA compute compatibility 3.5. It provides new shuffle instructions and grid management unit. At a low level, GK110 sees an additional instructions and operations to further improve performance. New shuffle instructions allow for threads within a warp to share data without going back to memory, making the process much quicker than the previous load/share/store method.  It also includes NVIDIA GPUDirect although it's RDMA functionality is reserved for Tesla technology only. Another interesting feature of this series is dynamic parallelism. Dynamic Parallelism allows kernels to be able to dispatch other kernels. With Fermi, only the CPU could dispatch a kernel, which incurs a certain amount of overhead by having to communicate back to the CPU.By giving kernels the ability to dispatch their own child kernels, GK110 can both save time by not having to go back to the CPU, and in the process free up the CPU to work on other tasks.[36]

## 2. 5 Matlab

Matlab (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Matlab allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Although Matlab is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. Matlab is widely used in academic and research institutions as well as industrial enterprises. Matlab can easily interface with other languages. Libraries written in Perl, Java, ActiveX or .NET can be directly called from Matlab, and many MATLAB libraries (for example XML or SQL support) are implemented as wrappers around Java or ActiveX libraries. Calling Matlab from Java is more complicated, but can be done with a Matlab toolbox or using an undocumented mechanism called JMI. Matlab can also call functions and subroutines written in the C programming language or Fortran. A wrapper function is created allowing Matlab data types to be passed and returned. The dynamically loadable object files created by compiling such functions are termed "MEX-files

# 3. Implementing AwSpPCA

## 3.1 Database

Tan, K.Songcan, C. [1] has introduced the algorithm that this paper intends to parallelize. Here, three separate databases are used to evaluate the efficiency of AwSpPCA; the ORL database, the Yale database and the AR database. Our thesis uses the Yale database to perform evaluations. The Yale Database contains two sets of faces – one set contains the subject's faces against a significant portion of the background, while the other has been cropped to only present the facial area. These faces are presented in different lighting conditions from different directions. It was chosen purposefully to make recognition harder and reveals the strength of Aw Sp PCA against PCA. The database contains faces of 39 individuals, both male and female, with 64 images per person making a total of 2496 images. The images are in 192x180 pixels, in the '.pgm' format.

## 3.2 Image Preprocessing

The cropped Yale database was used and the pictures were resized into 77x66 pixels. These dimensions were chosen because they provide a greater scope of Sub-patterning and the reduced pixels allows for faster calculations in covariance matrix and eigenvectors. However this resizing lowers the quality of the images and as such must be done with reserve. The implementation of Aw SP PCA requires the use of a Training Set and Test Set of images. Therefore, we needed to split the database into two sections. The Training sets and the Test sets were generated at random from the database. Depending on the implementation, up to 30 training images and 30 test images for every person were selected to run the algorithm on.

## 3.3Training Database

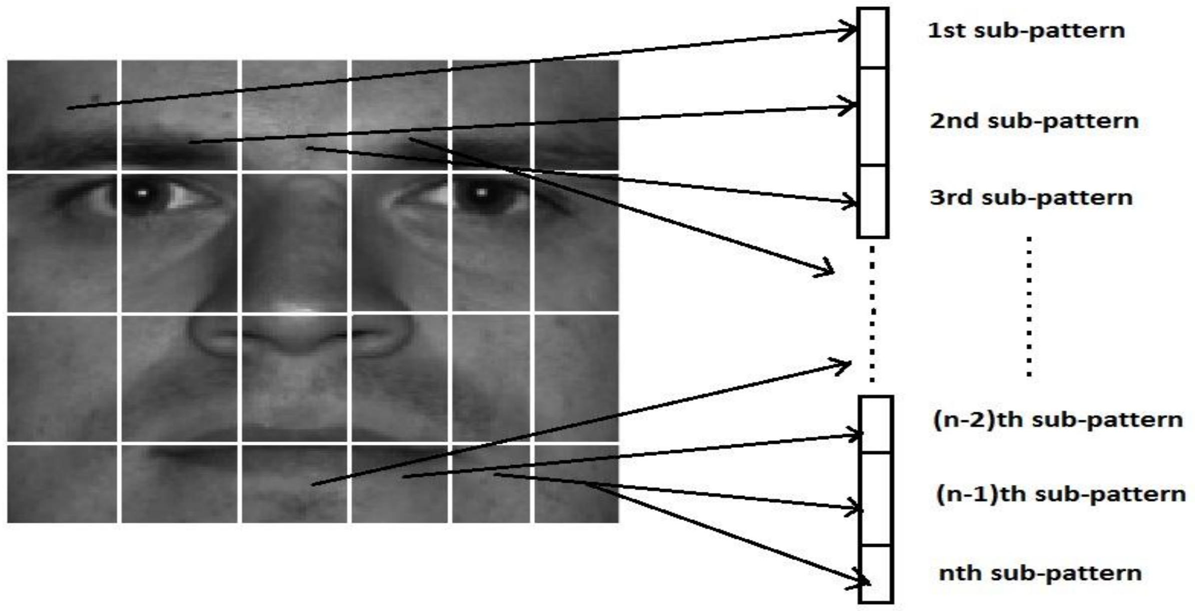
### 3.3.1 Sub-Pattern Partitioning

The Yale database is arranged as subfolders containing all the images of one person. There are 39 folders in total, and depending on the number of training images we had decided to train the database with, meaning a fixed number of random pictures from every folder is chosen as input. It is necessary to keep count of this for every person as will be shown later. The selected image is taken and partitioned into portions as selected and arranged into a one dimensional array containing the grayscale values of the pixels. Thus we have the vectorized representation of an image.

For image dimensions of ***N*** width and ***M*** height in pixels (totaling a ***N*M*** number of pixels per image), the image can be partitioned into numbers of rows which are a factor of ***M*** and column numbers which are a factor of ***N*** so as to avoid a loss of pixel data when creating sub-patterns. Therefore, for sub-patterns of ***m*** columns and ***n*** rows will give a total of ***m*n*** sub-patterns. The total number of pixels per sub-pattern will be given by

$$Sp = (N^*M)/(n^*m)$$

After being read, the image is cropped into a specified number of sub-patterns. The pixels are appended under each other in one column array for that sub-pattern. The next sub-pattern is converted into another 1-dimensional array and this is concatenated to the end of the previous sub-pattern's column array. This process is iterated over the remaining sub-patterns and eventually results in a vectorized array for one image. For the next image in the dataset, the above process is repeated and another column array is generated and so on, until all images for one person has been vectorized and placed next to each other. Thus, for ***I*** number of images for one person, we will have an array of ***N*M*** height and ***I*** width.



**<u>Figure 1. – Partition Image into Sub-Patterns</u>**

### 3.3.2 Probe Sets

After the vectorization of every training image of one person, we must create one column array for the mean and another column array for the median of the set of images. The mean is calculated by averaging the pixel values in every row of the vectorized training images calculated in the previous step. For the median, the middle values for the pixels are taken. Thus for all images for a single person, in the training set, we will have a rectangular array of the vectorized image array, a 1-dimensional column array for the mean and another for the median.

### 3.3.3 Filling up the database

Given a particular database, the algorithm assumes a number of images will be used as the training images and to generate the probes sets. This is done for a multitude of different people and thus the training dataset is completed. Therefore, given a set of images with ***J*** different people, and number of images of ***I*** per person, we should have ***J*I*** number of vectorized training images and ***J*** mean and median probe sets.

### 3.3.4 Parallelizing

The above steps are relatively simple and require little calculations on the Processing unit’s part. For the serialized version, the images are read in a loop and one image is taken per cycle. CUDA allows access to the processing capabilities of the NVIDIA GPU and gives the ability to run calculations on the GPU in parallel. The GPU used (See Chapter 4.1) contains 12 multiprocessor cores and they were used to perform computations in parallel. The results are shown in Figure 2. This section was chosen since the computations here can be considered embarrassingly parallel. Also, the next section of the algorithm requires data from every pixel in every image; otherwise the covariance cannot be calculated, which is why the next section could not be put in parallel with the execution of this particular instruction sequence. The sequence of the tasks being executed is such that the Single Instruction Multiple Data [3] system would work on multiple threads to process the data of one image per core. Upon completion of the input of one of the processes of any core, it will move on to the next available image that has yet to be read and re-executes the instruction sequence on that image.

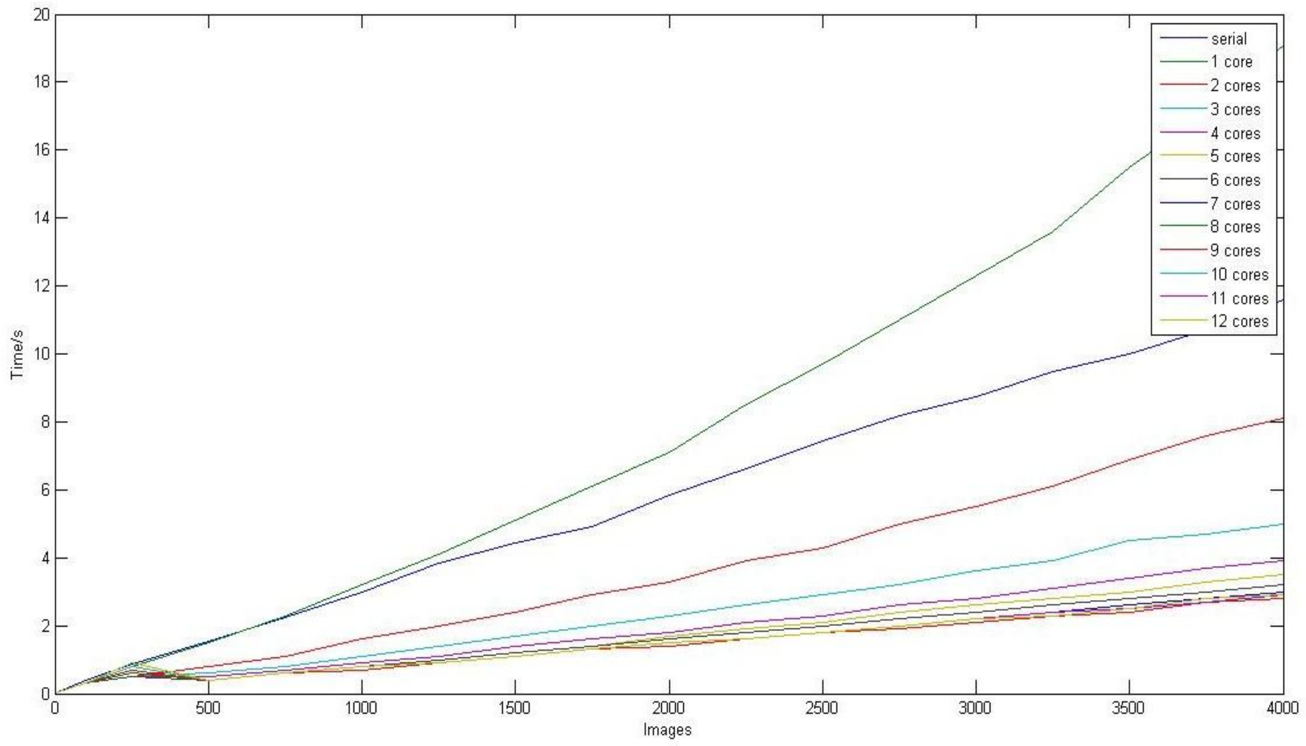


**Figure 2. Number of Input Imagesvs. Time**

The curves for the number of cores used grow linearly with number of image inputs. It can also be seen from the graph plot that while at inputs at less than 250 images, it is more efficient to use the serial implementation. However, an anomalous change is detected with the use of a single core of the GPU. As the number of inputs is increased, the time taken is also increased, but not linearly as is the case with the others. It increases with a certain exponential characteristic. This is because initializing threads and workers on a single processor has certain overheads which simply do not scale with the workload it has been presented with. The GPUs are designed for multitasking, therefore by limiting its work to a single GPU, much of the work it does is redundant and inefficient, resulting in the rise in time.

Note that while the graph functions were plot against all 12 cores, the last data for the last three cores are not presented since they show practically no improvement and the execution times. The reasons for this are explained in Chapter 4.4.

## 3.4 Principle Component Analysis

### 3.4.1 Calculating Covariance Matrix

The resultant of the previous steps produces a vectorized image array. Depending on the number of sub-patterns that has been decided on, this vectorized array is split for the calculations in the next steps. Given that the vectorized image array has ***N*M*** (row pixels and column pixels) pixels and the length of the array is ***J*I*** (total number of images), without sub-pattern partitioning we will require to calculate covariance with ***N*M*** values. The covariance is calculated by

$$q_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} \left( x_{ij} - \bar{x}_j \right) \left( x_{ik} - \bar{x}_k \right)$$

For details on the calculations for covariance, see [18].

Here, the covariance is calculated by summing the multiplied result of the mean subtracted row against another row and then dividing it by the total number of columns. The row contains the pixel values of all the images and therefore the covariance calculated here is the covariance between the pixels. With sub-patterns, we will create covariance separately for each of them, thus significantly reducing the time required to calculate them. With a larger number of sub-patterns, there will be lesser pixels per sub-pattern and therefore less time required to calculate them. Putting it all together, we can say that given ***S*p** being the number of pixels per sub-pattern, for every one sub-pattern the covariance matrix dimensions will be ***S*p****S*p.**

It must also be noted that the number of sub-patterns significantly affect the accuracy of the results. The details of the results can be found in Figure 3. The σ [Chapter 3.4.3] is kept at a constant 100% and the dataset for training and testing are kept the same while the sub-pattern row and sub-pattern columns are changed to check for their resulting differences in accuracy. It has been found that an increase in sub-pattern will increase accuracy, though not proportionally. These findings encourage the use of greater number of sub-patterns to optimize the Aw SP PCA algorithm both in execution time and in results.

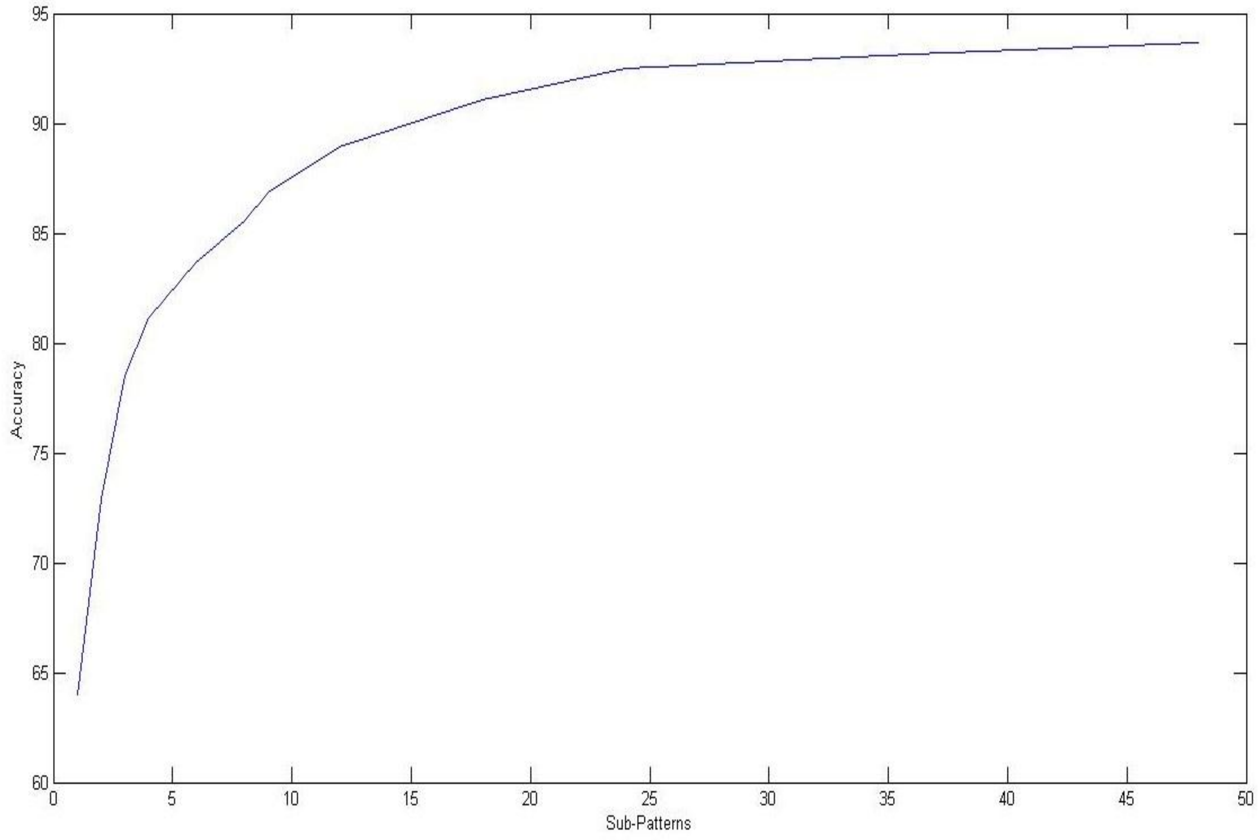


**Figure 3. Accuracy vs. Sub-patterns**

It is quite obvious that a greater number of sub-patterns yield better accuracy. However, noticing the trend with which the accuracy rises, it becomes apparent that after 35 sub-patterns, the accuracy falls victim to diminishing returns so it might seem redundant to use any more.

### 3.4.2 Eigenvectors and Eigenvalues

The covariance matrix is a representation of the relationship between the pixel values of the images. It can be said that the covariance can accurately show the deviation of every pixel against each other. Being a square matrix, the covariance matrix can have eigenvectors and eigenvalues derived from it. This process is equivalent to finding the axis system in which the covariance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of the greatest variation, the one with the second largest value is the (orthogonal) direction with the next highest variation and so on. Let A be a square matrix with k X k dimensions. The eigenvalues of ***A*** are defined as the roots of

$$\textit{determinant}(A - \lambda I) = |(A - \lambda I)| = 0$$

where I is the k Xk identity matrix. This equation is called the characteristic equation (or characteristic polynomial and has k roots.

Let $\boldsymbol{\lambda}$ be an eigenvalue of $\boldsymbol{A}$. then there exists a vector such that

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

The vector $\boldsymbol{x}$ is called an eigenvector of $\boldsymbol{A}$ associated with the eigenvalue $\lambda$. Notice that there is no unique solution for $\boldsymbol{x}$ in the above equation. It is a direction vector only and can be scaled to any magnitude. To find a numerical solution for x we need to set one of its key elements too an arbitrary value, which gives us a set of simultaneous equations to solve for the other elements. The values that result are usually normalized as $\boldsymbol{x^*x' = 1}$.

What the above procedure effectively does is come up with a new set of axes directions which better suit to represent the vectorized images. The eigenvectors are essentially the new directional axes upon which a projection of the images will give a set of coordinates on what is called the eigenspace. It is in this eigenspace that we calculate the Euclidean distances to come up with a sub-pattern weight set and classification results as shown in Chapter 4.

This is the second last step in performing Principle Component Analysis. The generation of the Eigenvectors and their corresponding eigenvalues makes it possible to find the underlying the components in a set of data, thus being called the Principle Components. As was seen in the previous section regarding the relationship between sub-pattern number and calculation times for covariance matrices, the same is true for eigenvector calculations. A smaller covariance matrix will yield a smaller eigenvector matrix because of the need for fewer computations and result in faster outputs.

### 3.4.3 Dimension Reduction Heuristics

The next step is to organize the eigenvectors in descending order with respect to their eigenvalues. In other words, the eigenvector with the highest eigenvalue is put first, the second highest is put next and so on. The purpose of doing this is to analyze the eigenvectors to see which ones have the most impact and which have the least on the dataset. In terms of this implementation, the results of sorting the eigenvalues will give us the pixels which have the least impact (meaning change the smallest) throughout the dataset. After sorting, it becomes apparent that certain pixels are more important than others on the variations of the pixel values in the dataset. The less important ones, if significantly low enough can be ignored and removed from the original eigenvectors generated, resulting in a smaller eigenvector matrix.The decision of which portions (or how much) of the eigenvectors is unnecessary requires a heuristic approach to it (one may even consider the usage of Artificial Intelligence algorithms to come up with an optimal solution). The mathematical approach to this is to assign a value σ, where

$$\boldsymbol{\sigma = (\textit{number of eigenvectors taken/total number of eigenvectors}) * 100}$$

By changing the value of σ, we can quite effectively manipulate the algorithm into choosing the top few percent of the total values. To figure out the changes, a graph of σ versus accuracy is plotted, and the results are show in Figure 4.

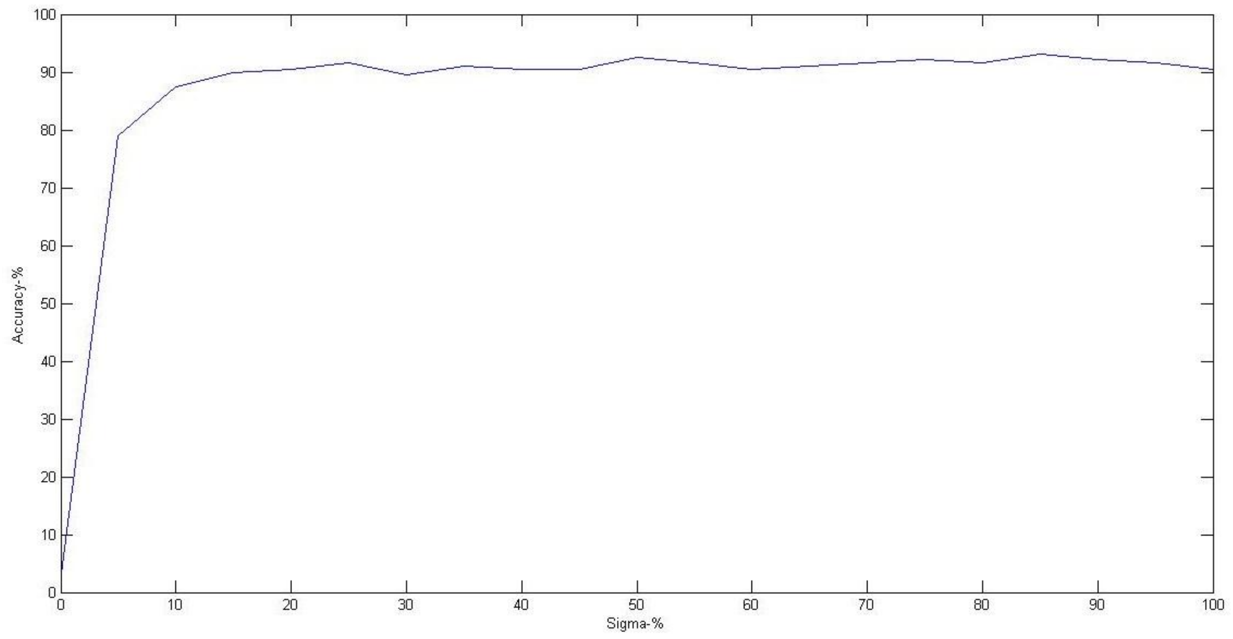


**Figure 4.Accuracy vs. σ**

From the above graph, it is apparent that increasing the σ over 10% results in lower increases in accuracy. The highest accuracy was at 50% at 92%.

### 3.4.4 Parallelizing

As mentioned above, the best way to improve the running time of the algorithm can be set by increasing the number of sub-patterns used. There are multiple prospects of applying parallelization to this portion of the algorithm. The first would be to select one particular sub-pattern from the vectorized images matrix, and send it to one core to perform eigenspace calculations and return that particular sub-pattern's array, with every core working on a different sub-pattern.The other option would be to take every sub-pattern serially and compute the covariance between two particular rows on one core and other combinations on other cores in parallel (it is not possible to parallel compute eigenspace due to its inherent requirement of data dependency). However this was not chosen as the parallel model due to the fact that covariance matrix has little computational costs with larger number of sub-patterns and contribute little to the overall time for this portion. It must be realized that GPUs specialize in executing mathematical functions efficiently and covariance calculations are just a series of summations and multiplications, making the calculations very efficient in a GPU. Therefore, the above model was chosen to be the parallel model.

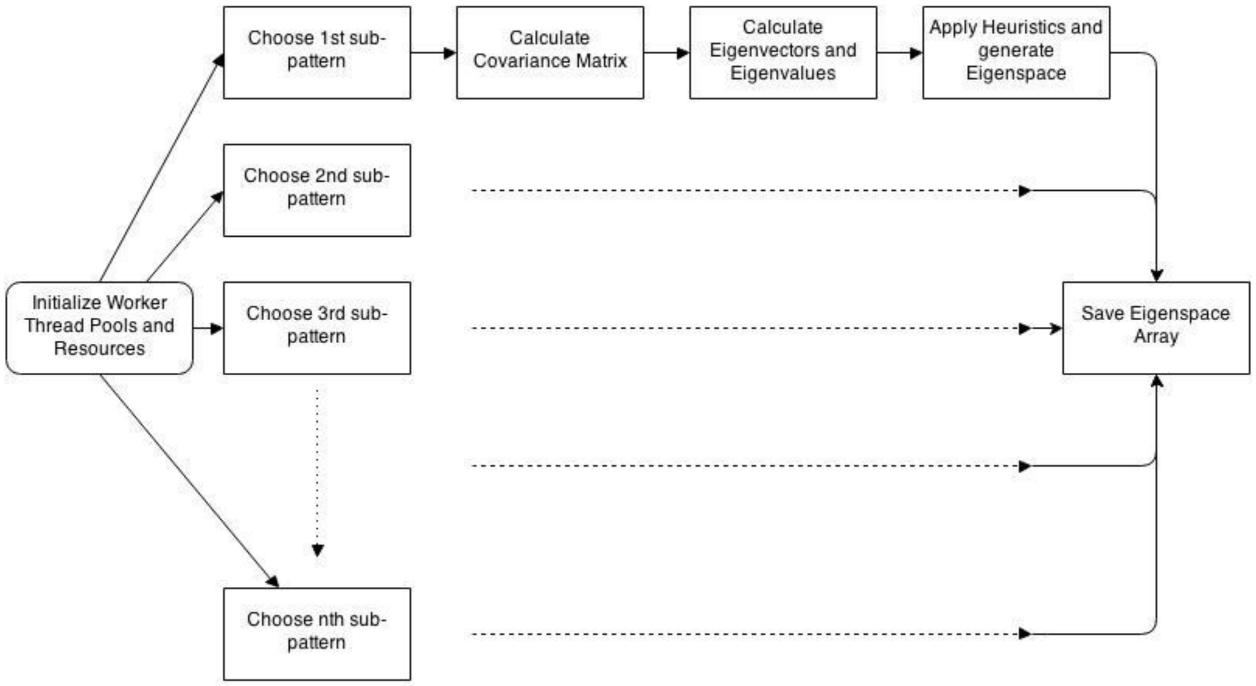

Initialize Worker Thread Pools and Resources
Choose 1st sub-pattern
Calculate Covariance Matrix
Calculate Eigenvectors and Eigenvalues
Apply Heuristics and generate Eigenspace
Choose 2nd sub-pattern
Choose 3rd sub-pattern
Choose nth sub-pattern
Save Eigenspace Array

**Figure 5 – Parallel implementation of Eigenspace generation**

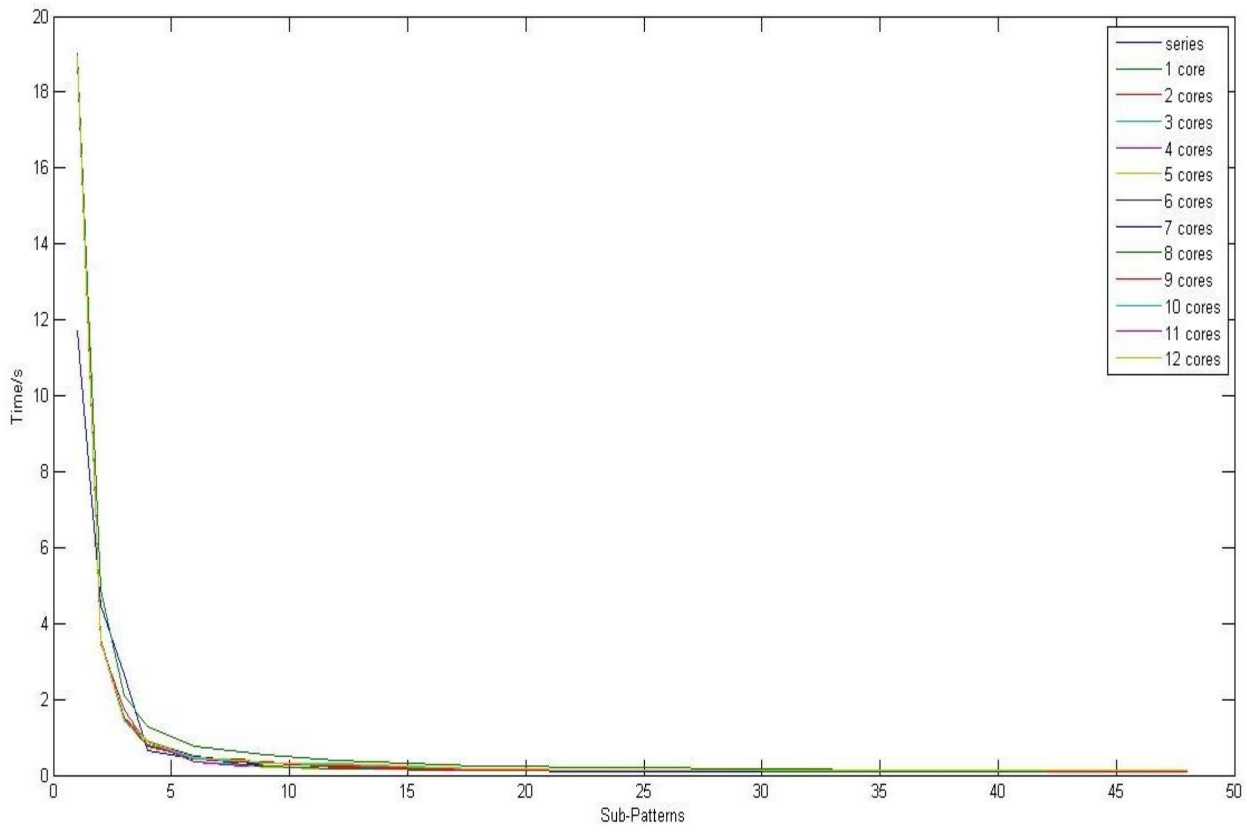


**Figure 6 – Time vs. Sub-pattern**

It can be seen that the steepest drops in time results from the increase of sub-patterns rather than the use of multiple cores of the GPU. There is an extreme improvement for sub-pattern numbers greater than 4, but any other numbers beyond that yield little benefits to execution time. This can be attributed to the fact that by partitioning an image, we are effectively lowering the number of dimensions that will be required for calculation, so eigenspace and covariance calculations show drastic improvements. However, the use of multiple cores shows practically no difference, which was a surprise since even the least amount of performance benefits could have been expected, however minimal. Since no performance benefits were apparent, it seems as though this section of the algorithm was better left in its serialized version

## 3.5 Computing Sub-pattern Weights

### 3.5.1 Weights

The major difference between classical Principle Component Analysis and Adaptively weighted Sub-patterned Principle Component Analysis is the latter's use of taking into account the separate contribution degrees of a particular portion of the image. These contributions are given a quantitative value in the form of weights. These weights are not assigned by the programmer. Instead, a separate function must be written in order to compute the weights per sub-pattern through the use of the mean and median (as calculated in Chapter 3.4.1) probe sets and by comparing them against the training set. The contributions of a sub-pattern are basically how important that particular portion of the image is in determining the final output of the classification result. The process of calculating a sub-pattern's weight starts by comparing that sub-patterns probe sets and matching them against the training set images. If the sub-pattern mean or median manages to get a correct hit, then the sub-pattern's weight is increased by one. In this way, all sub-patterns are checked for accuracy and their weights adjusted accordingly.

### 3.5.2 Projection on Eigenspace

Given a set of eigenvectors, a projection of a matrix or matrices means placing that matrix in the eigenspace in the form of a point on the coordinate system. This is done through matrix multiplication of the eigenvector matrices with the matrices to be placed. For the purposes of this algorithm, the set of vectorized images are the matrices that need to be projected in the eigenspace, as is given by –

***EigenSpaceCoordinates = Eigenvectors * VectorizedImagesMatrix***

As such, given two such points in the eigenspace, the distance between them can be measured in Euclidean Distances. The formula for it is given by –

$$Euc(x, y) = ((x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots\ldots + (x_i - y_i)^2)^{1/2}$$

### 3.5.3 Calculating Sub-pattern Weights

In classical Principle Component Analysis (for Facial Recognition), two images, the unknown image which needs to be classified and the database image against which it is being classified, are projected on the eigenspace and the distances are calculated through the Euclidean distance formula mentioned above. The image to be classified is compared against the distances of all the images in the training database and the training image which has the least distance is considered to be the closest match to the unknown image. The same is true for Aw SP PCA. However, instead of comparing whole images, sub-patterns are compared to their corresponding sub-patterns in other images. This is how the weights of the sub-patterns are calculated. The process begins by initializing an array of size equaling the number of sub-patterns. Next, the median and mean of one person is taken and one of the sub-patterns is selected from both the mean and median. This sub-pattern is then compared to the all the training set images' corresponding sub-patterns. The step sequence is shown in Figure 8. The results of the comparisons are checked. If the sub-pattern comparison yields the correct result, that is, if the closest Euclidean distance of the mean or median sub-pattern is the sub-pattern of one of the correct person's image to whom the probe set mean and median array belong to, then the value of that sub-pattern's weight array is increased by one. After the calculations for every sub-pattern has been completed for every mean and median array from the probe sets, then we will have the completed weight array. As a finishing touch, the values of the weights are recalculated to –

$$W_i = W_i \ /(2 * M)$$

Where $M$ is the number of means and medians in the probe set and $W_i$is the weight of sub-pattern $i$.

As can be seen from the steps, the iteration is threefold. The first loop iterates through all the probe set images, meaning once for every different person. The next loops through all the sub-patterns and the last one goes through every image in the training set. Therefore, given that we have $i$ number of people with every image being split to $j$ sub-patterns and a total of $m$ images in the training database, we will require $i*j*m$ iterations in total.

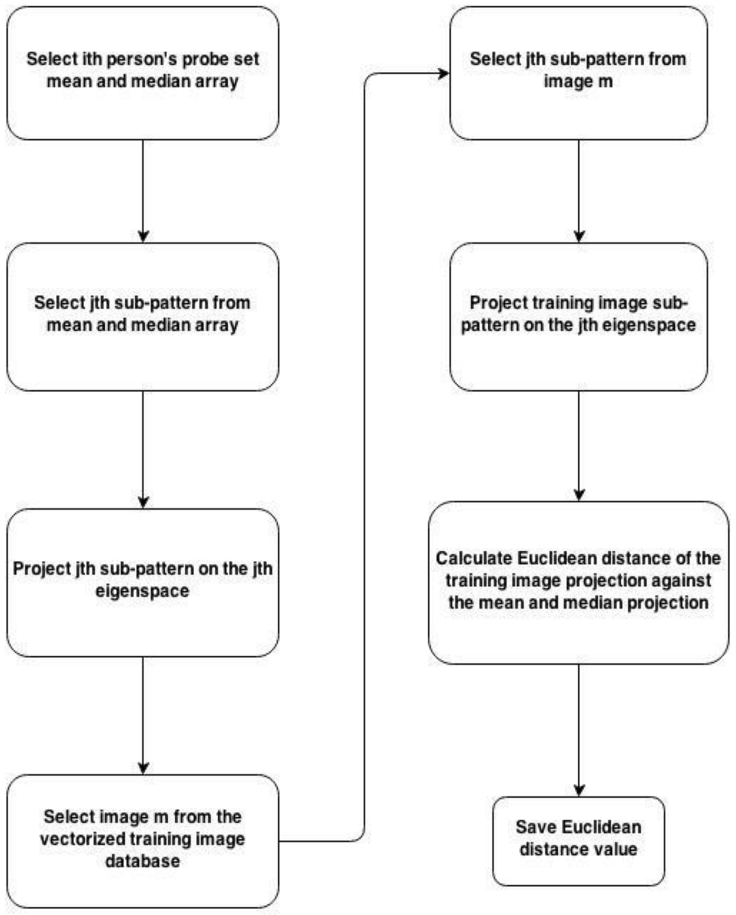

Select ith person's probe set mean and median array
Select jth sub-pattern from mean and median array
Project jth sub-pattern on the jth eigenspace
Select image m from the vectorized training image database
Select jth sub-pattern from image m
Project training image sub-pattern on the jth eigenspace
Calculate Euclidean distance of the training image projection against the mean and median projection
Save Euclidean distance value

**Figure 7 – Euclidean Distance Calculation**

### 3.5.4 Parallel Implementation

Given the number of iterations one must undergo here, and the fact that there is a great level of data independency per iteration, the parallel implementation prospects seem lucrative. The greatest number of iterations must be done in the loop that goes through all the training images. Thus it made most sense to perform one training image comparison per core. As mentioned before, the parallel architecture chosen was the Single Instruction Multiple Data structure. We have seen previously that the optimum number of sub-patterns is 36, which is why this number is kept constant while the Time vs. Cores is evaluated.

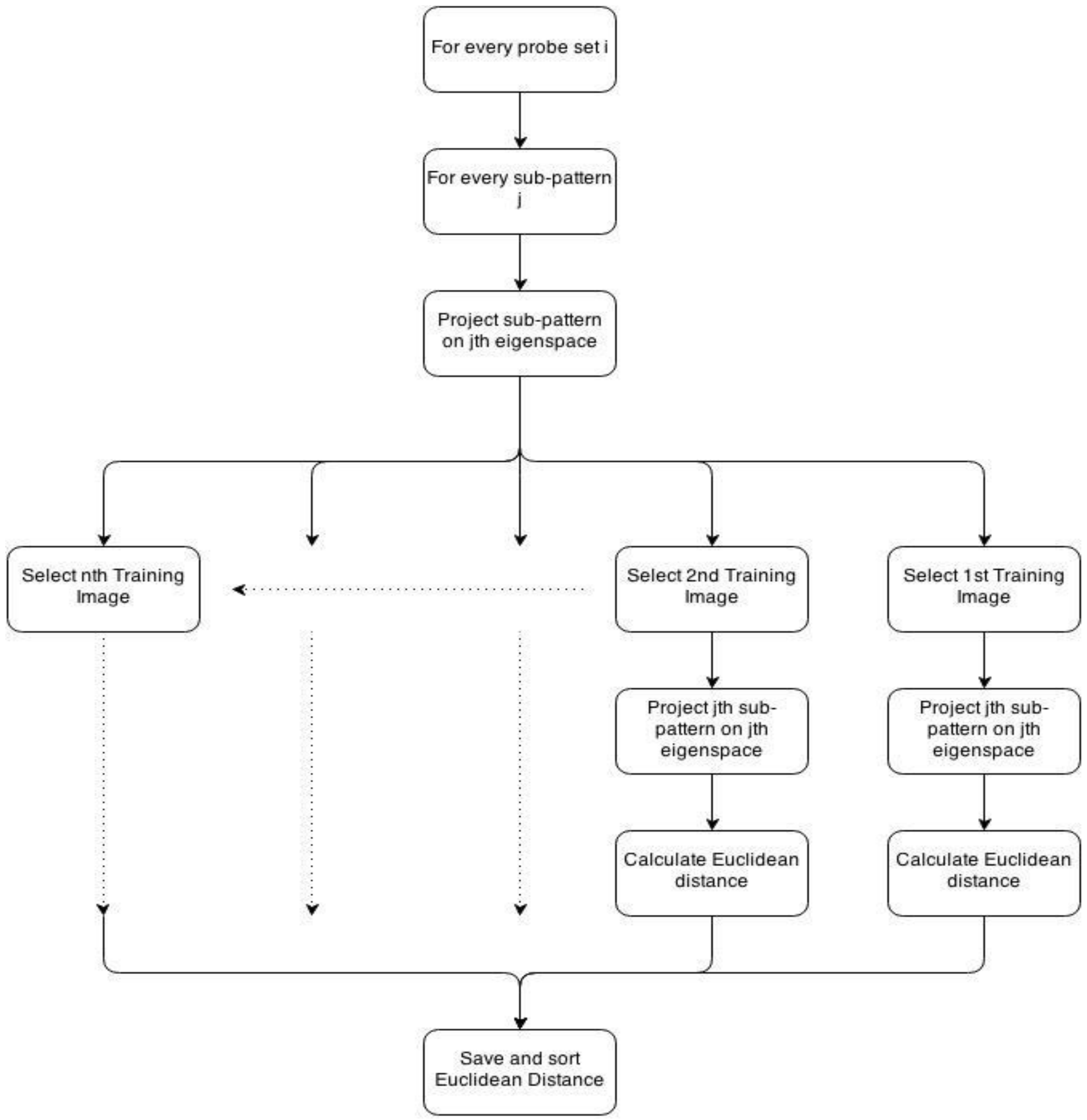

For every probe set i
For every sub-pattern j
Project sub-pattern on jth eigenspace
Select nth Training Image
Select 2nd Training Image
Select 1st Training Image
Project jth sub-pattern on jth eigenspace
Project jth sub-pattern on jth eigenspace
Calculate Euclidean distance
Calculate Euclidean distance
Save and sort Euclidean Distance

**Figure 8. – Parallel Implementation for sub-pattern weights**

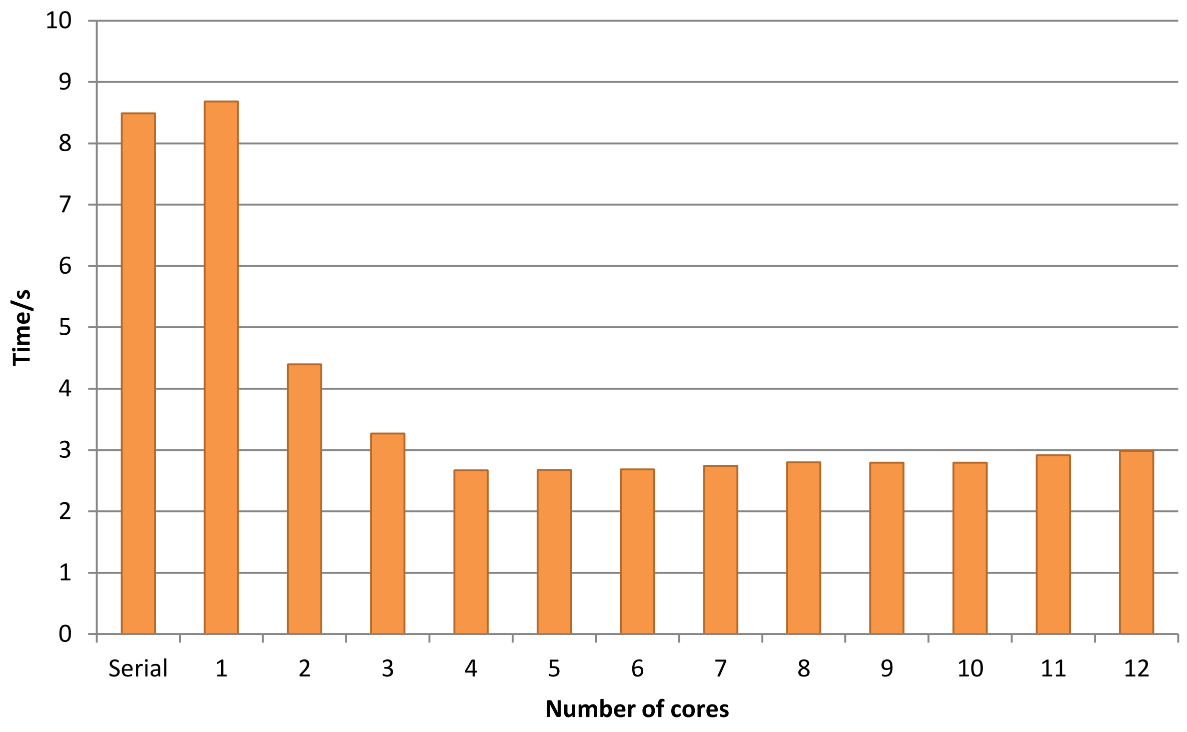


**Figure 9. – Sub-pattern Weights - Time vs. Cores**

While it is expected that there should be a massive level of improvement for all cores, it is apparent that thread controlling overheads tend to take a toll on increase of cores. Also, there is a notable rise in execution time on using a single core of the GPU than using the series implementation on the CPU. This can be explained by the fact that the CPU used an Intel Core i7 processor, which is a significantly powerful commercial processor with fully utilized 8 cores. Also, by limiting the number of cores in the GPU by 1, we are significantly increasing the tread overheads, resulting in the rise in time.

## 3.6 Classification

### 3.6.1 Test Image Vectorization

The test images dataset must also be vectorized similarly as the training set images in Section 3.3. It must be noted that the test images must also have the same dimensions as the training database images and thus be subjected to the same degree of partitioning. Ultimately the test image dataset will be created with the same processes just with the lack of creating a probe set for a set of images.

### 3.6.2 Classification

The first step of this stage is to initialize a total weights array with a size equal to the number of people there was in the training database. This array holds the cumulative weight as calculated by the recognition algorithm. It is almost the same as the previous section, since it also requires measuring the Euclidean distances between sub-patterns.

The steps start with selecting a test image from the vectorized images set. This image is then broken into its sub-patterned sections and projected on the corresponding eigenspace. The next step is to iteratively take every training database image's same section sub-pattern and project it on the same eigenspace. The Euclidean distance is measured and saved. After all the images have been measured against, the closest image to the test image is picked, and that image's corresponding person's weight is increased by the amount in that sub-pattern's weight array (the result from Section 3.5).

After all training images have been looped through, the algorithm moves on to the next sub-pattern and the whole process is repeated for one test image until all sub-patterns have been checked. The total weight array is then sorted, and the person with the greatest weight is considered the closest match to the test image and is given as a 'hit'.

### 3.6.3 Parallel Implementation

The image vectorization can be subject to instruction parallelism, with the performance graphs and implementations being virtually the same to Figure 2.

Classification goes through the familiar steps from Section 3.5, with the differences being in the calculation of weightings and the inclusion of test input images. The iterations start with selecting all test images, then looping through all the sub-patterns. Lastly, all training images are gone through, which ultimately results in an algorithmic structure similar to those taken to calculate sub-pattern weight. Thus, it can also be subject to the parallel structure discussed above in Figure 9.

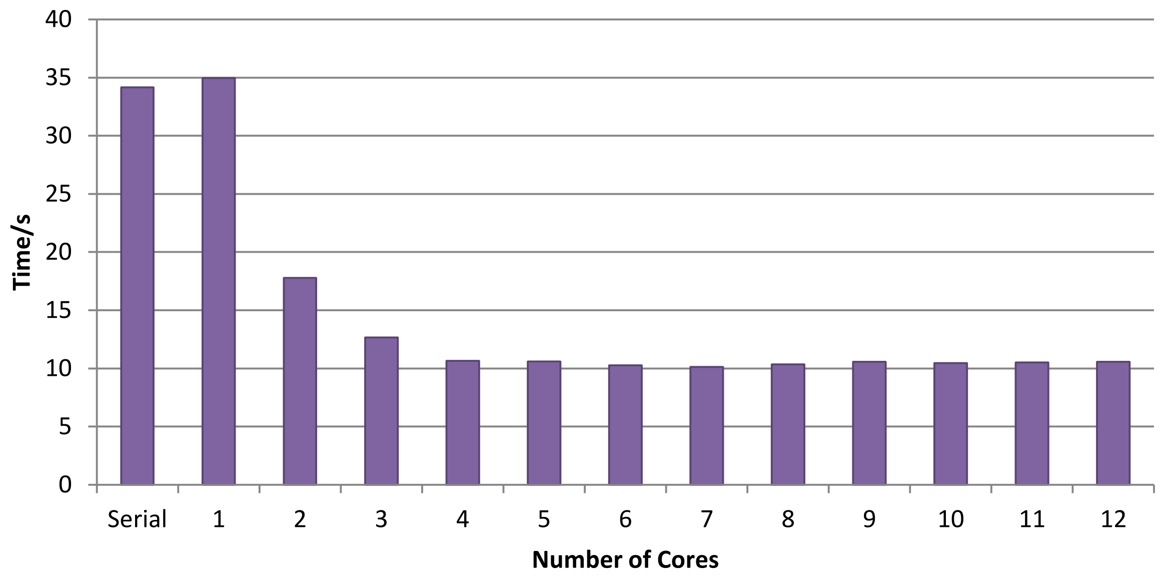


**Figure 10.Classification - Time vs. Cores**

The explanations for this section are the same as in Chapter 3.5.4. However, one may note the significant differences in the values from Figure 10 in that this time the lowest runtime is for the 7 core. Due to the greater number of computations involved in this part of the algorithm, it is apparent that it can fully utilize the processing capabilities of the 7 core processor, without being encumbered by overheads. It has been shown in [41] that larger problem sizes give better efficiency with a higher number of processors. So this section, having to undergo more iterations due to larger data size, is more efficient at the use of more cores.

# 4. Results and Discussion

## 4.1 System Environment

PC Specifications

- Windows 8 – 64 bit
- CUDA – Version 3.5
- MATLAB 2013b

PC 1 Hardware

- Intel Core i7 4770 3.40GHz
- Number of cores in CPU: 8 cores
- Ram 16 GB
- GPU: NVidia GTX 780
- Multiprocessor Count for GPU: 12
- Number of CUDA core in GPU: 2304
- GPU memory size: 3GB
- GPU memory Interface width: 384-bit

PC 2 Hardware

- Intel Core i7 4770 3.40GHz.
- Number of cores in CPU: 8 cores.
- Ram 16 GB.
- GPU: NVidia GTX 660
- Multiprocessor Count for GPU: 6.
- Number of CUDA core in GPU: 960.
- GPU memory size: 1GB.
- GPU memory Interface width: 192-bit.

## 4.2 Accuracy

From Figure-3, it can be seen that there were no significant performance benefits in runtime after increasing sub-pattern size over 4, therefore the accuracy benefits over any sub-pattern size over 4 should be considered. Considering the results from Figure 3, sub-pattern size 35 seems to give the best results in accuracy. Since the number of sub-patterns must be the multiple of the factors for the image dimensions, the final number of sub-patterns chosen was 36.Similarly, from Figure 7.itcan be concluded that by taking a **σ** value of over 30% will not result in significant increases in accuracy, and the peak accuracy is reached at a value of 50% (92% accuracy for 36 sub-patterns). While a greater number of **σ** will result in a greater time for the overall execution, it must be understood that a compromise must be made between speed and accuracy. An increase of accuracy from using **σ** values from 30% to 50% results in an increase of 3% in accuracy, while any more increase does not provide any more benefits, but has greater execution time. Therefore, the **σ** of 50% was chosen, culminating to a final accuracy of 92.6315%.

## 4.3 Results

Going by the data from Figures 2, 3, 4, 6, 9, and 10, a reasonable estimate can be made on the time taken for the complete execution of the algorithm for every core used. The analysis proves that the best cores to be used would be 4 cores (for Sub-pattern Weights calculation) and 7 cores (for Recognition and Classification). It would have been best to be able change the number of cores for every section, but the time taken to pool resources and redistribute them again when changing cores requires too much time where the CPU will be involved in unnecessary work, so a compromise needs to be made, because the average number, which is 5, is taken.

The use of 7 cores in the classification phase as compared to using 5 cores has a difference of approximately 0.47s. Similarly, the use of 4 cores in the Sub-pattern weights calculation phase as compared to using 5 cores has a difference of approximately 0.01s. Thus a compromise of 0.08 seconds must be made by running the full algorithm on 5 cores of the NVidia GPU.

Next, the vectorization sections are also analyzed and for 570 image (the size of the training dataset) from Figure 2 and using 5 cores give a respectable difference in execution time compared to serial.

Based on these observations, it can be safely assumed that using 5 cores, with **σ**= 50% and sub-patterns = 36, should give us the optimum result.

Executing the algorithm on the above specifications on PC 1 gives an execution time of 12.6707s.

Similarly, the algorithm was re-analyzed for PC 2, and it has been found that the optimum number of cores for this machine was 4, with it giving a runtime of 14.2245s. The summary of the results are given below.

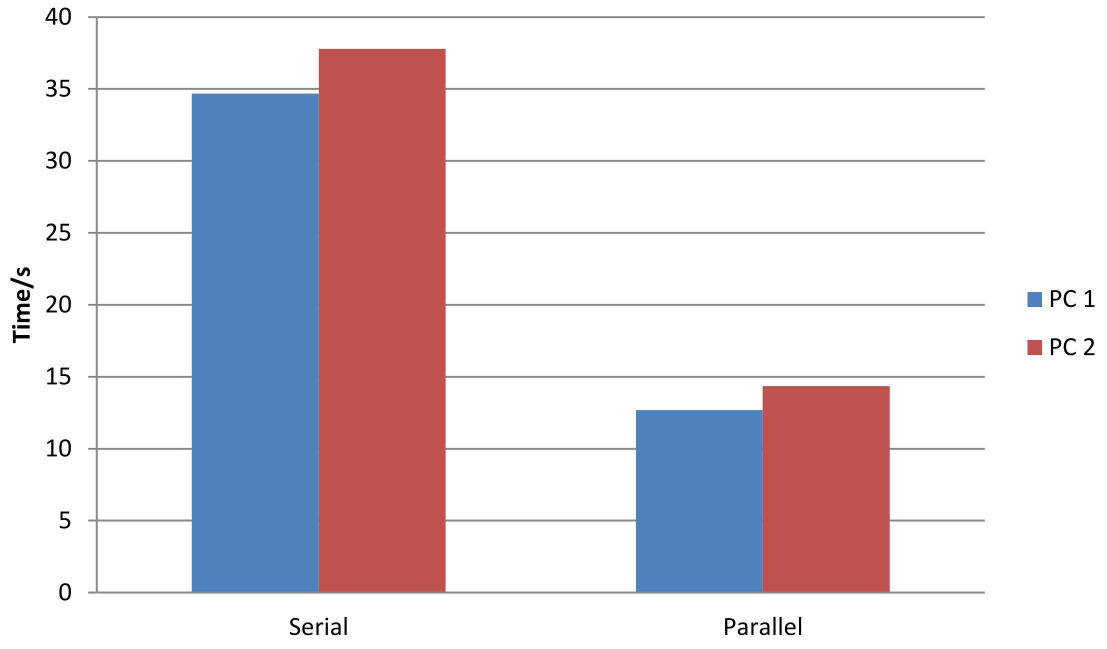


**Figure 11. – Aw SP PCA Sequential and Parallel Times**

## 4.3 Discussion

In order to further analyze the performance benefits, the speedup and efficiency of the speedup are calculated. They are given by –

$$Speedup = \frac{Sequential\ Execution\ Time}{Parallel\ Execution\ Time}$$

$$Efficiency = \frac{Speedup}{No.of Processors} * 100$$

| PC 1 | PC 2 |
|---|---|
| **Speedup = 2.74** | **Speedup = 2.62** |
| **Efficiency = 55.3%** | **Efficiency = 65.6%** |

**Table 1. Performance Analysis Results**

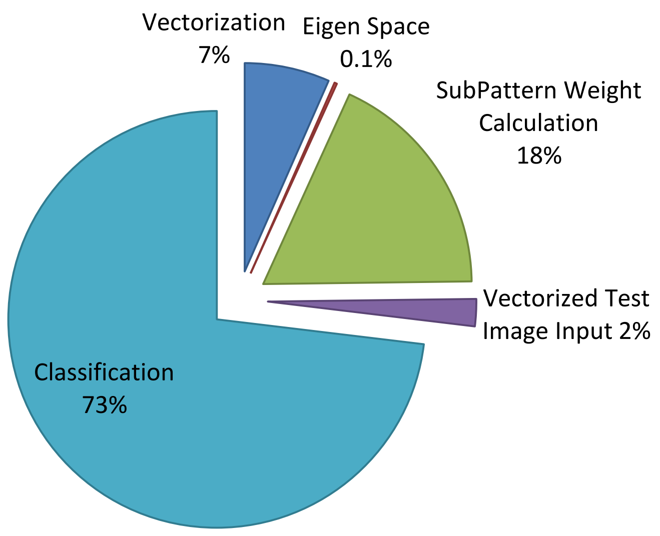


**Figure 12. Aw SP PCA ExecutionTime Distribution**

The time distribution of the Aw SP PCA Algorithm was measured through timing the different sections of the sequential implementation of the algorithm. According to Amdahl's Law, the maximum speedup of an algorithm is given by –

$$Maximum\ Speedup = \frac{1}{f + (1 - f)/N}$$

where $N$ = number of processors and $f$= % of algorithm inherently sequential

The version of the algorithm implemented has been calculated to have ***f*** = 8%. The number of processors being used for the final execution was 5 (for PC 1). Thus the calculated **Maximum Speedup** = 3.78.

Theoretically, the ideal time taken for execution against a number of processors should decrease proportionally. Had this been the case, the efficiency of speedup for Aw SP PCA would have been close to 100% and the speedup would have been close to the value calculated. However, practically it is much lower due to the cost constraint suffered by every parallel implementation of a program. These constraints are generally called “communication overheads” and are known to increase as per equation –

$$Algorithmic\ Cost = Parallel\ Runtime\ Cost\ \times\ Number\ Of\ Processors \quad [40]$$

The cost rises multiplicatively with additional processors, which explains the steady time taken after a certain number of processors are used. The problem size is also a factor. The explanations for these costs can be explained through detailed study of Load Balancing, Shared Memory, Resource Paging, Memory passing and more, with the details found in [21].

# 5. Conclusion and Future Work

## 5.1 Conclusion

Through the use of the Single Instruction Multiple Data design, the algorithm structure was changed to make the most efficient use of the parallel processing capabilities offered by CUDA and NVidia GPUs. The results are below the mark by quite a large margin when compared against the speedup efficiency and the expected values expected from speedup judging by Amdahl's Law. However, the aim of the paper, which was to implement parallel processing techniques to decrease the runtime was achieved to some degree. A speedup of over 2.5 is a significant improvement. The analysis of sub-patterns against and heuristics against accuracy also yielded beneficial results compared to [1]. An increase in accuracy of almost 5% was achieved.

## 5.2 Future Work

There remains much to be done about Aw SP PCA considering increasing its accuracy. One such work would be to analyze the benefits of scaling sub-patterns to see their effects on accuracy. Similarly, using the RGB values instead of using plain grayscale values to increase accuracy would also be an option. While this paper uses the SIMD to parallely implement Aw SP PCA, using the pipeline processing for Aw SP PCA to reduce runtime was another path that can be taken. Amdahl's effect, which says that parallelizing scalability increases with data size can be illustrated through the use of a larger database.

# 6. References

[1] Keren Tan,Songcan Chen, “Adaptively Weighted Sub-pattern PCA for Face Recognition”, Nanjing University of Aeronautics & Astronautics,Nanjing, 210016, China

[2] Bruce A. Draper, KyungimBaek, Marian Stewart Bartlettand J. Ross Beveridge,, “Recognizing faces with PCA and ICA”, Computer Vision and Image Understanding 91 (2003) 115–137
www.elsevier.com/locate/cviu 11 February 2003

[3]Blaise Barney, Lawrence Livermore National Laboratory, Introduction to Parallel Computing https://computing.llnl.gov/tutorials/parallel_comp/

[4]Delac K., Grgic M., Grgic S.: “Independent Comparative Study of PCA, ICA, and LDA on theFERET Data Set.” International Journal of Imaging Systems and Technology, Vol. 15, No. 5, 252–260, (2005)

[5]Rathore N., Chaubey D., Rajput N.: A Survey on Face Detection and Recognition. InternationalJournal of Computer Architecture and Mobility, Vol. 1, Issue 5, (2013)

[6]Dandotiya D., Gupta R., Dhakad S., Tayal Y.: A Survey paper on Biometric based Face DetectionTechniques. International Journal of Software and Web Sciences, Vol. 2, Issue 4, 67-76, (2013)

[7] Hjelmas E.: Face Detection: A Survey. International Journal of Computer Vision and Image Understanding, Vol. 83, 236-274, (2001)

[8] Smith L.: A tutorial on Principal Components Analysis. (2002). [www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf]

[9]Jafri R., Arabnia H. R.: Survey of Face Recognition Techniques. Journal of Information ProcessingSystems, Vol. 5, No. 2, 41–68, (2009)

[10] Turk T., Pentland A.: Eigenfaces for Recognition. Journal of Cognitive Neuroscience, Vol. 3, 71–86, (1991)

[11]Gao J., Fan L., Xu L.: Solving the face recognition problem using QR factorization. Journal ofWSEAS Transactions on Mathematics, Vol. 11, No. 1, 728–737, (2012)

[12]Goodall T., Gibson S., Smith M. C.: Parallelizing Principal Component Analysis for Robust FacialRecognition using CUDA. In 2012 Symposium on Application Accelerators in High PerformanceComputing, 121–124, (2012)

[13]Kshirsagar V. P., Baviskar M. R. Gaikwad M. E.: Face Recognition Using Eigenfaces. In 2011 3rdInternational Conference on Computer Research and Development, 5159–5164, (2011)

[14]Zhou C., Wang L., Zhang Q., Wei X.: Face Recognition base on PCA Image Reconstruction andLDA. Journal of Optik. Vol. 124, No. 22, 5599-5603, (2013)

[15] Draper B. A., Baek K., Bartlett M. S., Beveridge J. R.: Recognizing faces with PCA and ICA.International Journal of Computer Vision and Image Understanding, Vol. 91, 125-137, (2003)

[16] Wang W., Wang W.: Face Feature Optimization Based on PCA. Journal of Convergence Information Technology, Vol. 8, 693–700, (2013)

[17] Min L., Bo L., Bin W.: Comparison of Face Recognition based on PCA and 2DPCA. Journal ofAdvances in information Sciences and Service Sciences, Vol. 5, 545–553

[18]A tutorial on Principal Components Analysis,  Lindsay I. Smith February 26, 2002

[19]Amdahl's Law in the Multicore Era, Mark D. Hill and Michael R. Marty

[20] KanokmonRujirakula, Chakchai So-Ina, and BancharArnonkijpanichba, PEM-PCA: A Parallel Expectation-MaximizationPCA Face Recognition Architecture, Applied Network Technology (ANT) Laboratory, Department of Computer Science, Faculty of Science.

[21] Effects of Communication Latency, Overhead, andBandwidth in a Cluster Architecture
Richard P. Martin, Amin M. Vahdat, David E. Culler and Thomas E. AndersonComputer Science DivisionUniversity of California, Berkeley

[22]Abdi. H., & Williams, L.J. (2010). "Principal component analysis.". Wiley Interdisciplinary Reviews: Computational Statistics, 2: 433–459. doi:10.1002/wics.101.

[23]Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN0-8053-0177-1.

[24]S.V. Adve et al. (November 2008). "Parallel Computing Research at Illinois: The UPCRC Agenda" (PDF).Parallel@Illinois, University of Illinois at Urbana-Champaign.

[25]Barney, Blaise. "Introduction to Parallel Computing".Lawrence Livermore National Laboratory.Retrieved 2007-11-09.

[26] Brooks, Frederick P. (1996). The mythical man month essays on software engineering (Anniversary ed., repr. with corr., 5. [Dr.] ed.). Reading, Mass. [u.a.]: Addison-Wesley. ISBN0-201-83595-9.

[27] "Computer Vision Signal Processing on Graphics Processing Units", Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004): Montreal, Quebec, Canada, 17–21 May 2004, pp. V-93 – V-96

[28] Boggan, Sha'Kia and Daniel M. Pressel (August 2007). GPUs: An Emerging Platform for General-Purpose Computation (PDF). ARL-SR-154, U.S. Army Research Lab. Retrieved on November 7, 2007.

[29] "Facial Recognition Applications". Animetrics.Retrieved 2008-06-04.

[30] Crawford, Mark. "Facial recognition progress report".*SPIE Newsroom*.Retrieved 2011-10-06.

[31] Bonsor, K. "How Facial Recognition Systems Work". Retrieved 2008-06-02.

[32] Williams, Mark. "Better Face-Recognition Software". Retrieved 2008-06-02.

[33] Krause, Mike (2002-01-14). "Is face recognition just high-tech snake oil?".*Enter Stage Right*. ISSN1488-1756. Retrieved 2011-06-30.

[34] NVIDIA CUDA Home Page

[35] Silberstein, Mark; Schuster, Assaf; Geiger, Dan; Patney, Anjul; Owens, John D. (2008). "Efficient computation of sum-products on GPUs through software-managed cache".Proceedings of the 22nd annual international conference on Supercomputing - ICS '08. pp. 309–318. doi:10.1145/1375527.1375572. ISBN 978-1-60558-158-3.

[36] "CUDA-Enabled Products". *CUDA Zone*.Nvidia Corporation. Retrieved 2008-11-03.

[37] "NVIDIA Launches Tesla K20 & K20X: GK110 Arrives At Last". AnandTech. 11/12/2012.

[40] Performance Analysis, *www.cs.kent.edu/~jbaker/PDC-F07/slides/Chpt7.ppt*

[41] The Yale Face Database, vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html