# PARALLEL IMAGE PROCESSING FOR HIGH CONTENT SCREENING DATA

# PARALLEL IMAGE PROCESSING FOR HIGH CONTENT SCREENING DATA

By

TAMNUN-E-MURSALIN, M.Sc., B.Sc.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree

Master of Science

McMaster University

| | |
|---|---|
| MASTER OF APPLIED SCIENCE (2013) | McMaster University |
| (Biomedical Engineering) | Hamilton, Ontario |

TITLE: PARALLEL IMAGE PROCESSING FOR HIGH CONTENT SCREENING DATA

AUTHOR: Tamnun-E-Mursalin, M.Sc. (Northeastern University), B.Sc. (Northeastern University)

SUPERVISORS:

Associate Professor Dr. Qiyin Fang
Biomedical Engineering Department
Canada Research Chair in Biophotonics

Professor Dr. M. Jamal Deen
Electrical and Computer Engineering Department
Senior Canada Research Chair in Information Technology

Professor Dr. David W. Andrews
Biochemistry and Biomedical Sciences Department
Canada Research Chair in Membrane Biogenesis

Associate Professor Dr. Aleksandar Jeremic
Electrical and Computer Engineering Department

PAGES: vii, 84

## Abstract

High-content screening (HCS) produces an immense amount of data, often on the scale of Terabytes. This requires considerable processing power resulting in long analysis time. As a result, HCS with a single-core processor system is an inefficient option because it takes a huge amount of time, storage and processing power. The situation is even worse because most of the image processing software is developed in high-level languages which make customization, flexibility and multi-processing features very challenging. Therefore, the goal of the project is to develop a multithreading model in C language. This model will be used to extract subcellular localization features, such as threshold adjacency statistics (TAS) from the HCS data. The first step of the research was to identify an appropriate dye for use in staining the MCF-7 cell line. The cell line has been treated with staurosporin kinase inhibitor, which can provide important physiological and morphological imaging information. The process of identifying a suitable dye involves treating cells with different dye options, capturing the fluorescent images of the treated cells with the Opera microscope, and analyzing the imaging properties of the stained cells. Several dyes were tested, and the most suitable dye to stain the cellular membrane was determined to be Di4-Anepps. The second part of the thesis was to design and develop a parallel program in C that can extract TAS features from the stained cellular images. The program reads the input cell images captured by Opera microscopes, converts it to TIFF format from the proprietary Opera format, identifies the region-of-interest contours of each cell, and computes the TAS features. A significant increase in speed in the order of four fold was obtained using the customized program. Different scalability tests using the developed software were compared against software developed in Acapella scripting language. The result of the test shows that the computational time is proportional to number of cells in the image and is inversely proportional to number of cores in a processor.

# Acknowledgements

I cannot find words to express my sincere gratitude to my four supervisors.  Without their encouragement and patience this thesis would not have been possible. My deepest appreciation and thanks to Dr. Qiyin Fang for his sincere assistance and valuable feedback.   I am indebted for his constant assistance, encouragement, guidance and opportunities he provided throughout my graduate studies.  Thanks to **Dr. David Andrew** for introducing me to a new discipline of cell biology and allowing me to explore.  Thanks to **Dr. Aleksandar Jeremic** for his valuable guidance on image processing and useful suggestions on programming problems.  Thanks to **Dr. Jamal Deen** for his guidance and mentorship.

I would like to express my deep gratitude to Dr. David Andrew's Lab members for helping and allowing me to run the experiments in the facilities.  My sincere gratitude I would like to express my sincere appreciation to Dr. Ognian Marinov for his valuable and constructive suggestion during the writing of this thesis.  I am very grateful to Anthony Tsikouras for proofreading my thesis and for providing feedback.  Also thanks to Michael Nelson for proofreading.

Also thanks to the students and researchers in Dr. Fang and Dr. Andrew's Lab.  They were a great team to work with and I have learned many things from them.  More specifically, I thank to Fei Gang for hands on training on laboratory experiments from cell culture to cell treatment, to Jarkko Ylanko and Caitlin Mills for teaching me to use the Opera, serial dilution and for all the technical assistance.

Finally, I would like to thank all members of my family.  I start with mother for her unconditional love throughout my life.  It is impossible to put in words the thanks and appreciation of my wife and two kids.  Together, they have always been there for me, and I am forever in their debt.

This work is dedicated to my Mother, Professor Touhida Faruki Begum.  She has been a constant source of inspiration in my life.  She taught me the value of education, hard work and sacrifice.  Without these qualities, I would not have been able to achieve all that I have so far.  Thanks to her for all the support, inspiration and sacrifices.

## TABLE OF CONTENTS

# ***LIST OF FIGURES***

# ***LIST OF TABLES***

# 1 Introduction

## 1.1 Motivation

Since its discovery 10 years ago, information obtained from high content screening (HCS) has considerably advanced the field of drug discovery (Bickle, 2010). HCS has been well practised in all aspects of the drug discovery pipeline. It was widely used in primary screening, RNAI technology, toxicity analysis, and lead optimization (Rausch, 2006). Imaging in HCS has stimulated the advancement of two different fields: improvement of the hardware in automated microscopes and the enhancement of feature extraction software for image analysis (Zanella, Lorens, & Link, 2010). A standard screening would require several steps: incubation of cells in 96 or 384 well plates, treatment of cells with chemical compounds, staining with relevant fluorophores or tagged proteins, imaging each well by state of the art microscopes, and then finally interpretation of the images with quantifiable measurements. The root of the screening is the image analysis process where millions of multi parameter features are extracted. Examples of these extracted features are: texture, morphology, intensity and spatial distribution. However, with the advancement of HCS, there are still numerous challenges within the field of image computing, particularly in image processing, data mining, and visualization (Peng, 2008).

Even with huge development in high content screening analysis, there is still a desperate need to overcome the current hardware and software limitations (Starkuviene & Pepperkok, 2007). Rapid technical development in the field of fluorescence microscopy has enabled researchers to collect massive amounts of data; however, more processing power and data modeling tools would be required. Futhermore, with advancements in the fields of multispectral fluorophores, quantum dots, and fluorescent proteins: different cellular phenotypes of a cell can be measured with various parameters, which demand intensive computation, requiring more advanced algorithms and advanced hardware technology. For example, in a simple screening experiment a cell by cell image processing analysis of multiple spectra would require a huge computational power and time to process. However, due to the limitations on existing analysis software in the context of speed, the averaged population data was commonly used; this lacked detailed biological information of the phenotypes(Levsky& Singer, 2003). Current image analysis tools lack compatibility and integration, which consequentially failed to carry on with the high demand on processing of screening data (Wong, 2006); hence the benefit of HSC has not been fully exploited. Nevertheless, as the acquired data grows, the mining of biological knowledge has surpassed the capability of image processing tools. Therefore a different approach, such as parallel processing, would be needed to reduce the gap between current biological advancement and technological drawbacks.

## 1.2 High Content Screening: From Microscopy Imaging To Machine Learning

High content screening can be defined as a sequence of task flows where collective compounds are tested to observe the biological activities of living cell. Antje et al. have defined high content screening as phenotypic screening of cells on a multi well plate that uses tasks involved with automated microscopy, followed by automated image analysis, and analysis of many numerous numerical features. Thus, understating the workflow of high content screening is very significant.  Every stage of the screening pipeline is very important and must be performed carefully, following proper protocol and understanding of the automation system through each step.  Successful completion of one step is highly dependent on the previous successful step; therefore any error in any stage of the pipeline will affect the image analysis process. However, the screening task flow could also be designed according to the experiments and the biological questions; hence, number of factors needs to be considered and optimized (navigating).  Despite the flexibility of screening workflow, a typical high content screening experiment can be divided into the five following steps:

### 1.2.1 Assay Development

Assay development is required step of the High Content Analysis process. The first part of the assay involves selecting the cell lines for the experiments, which mainly depends on the particular research area of interest.  Once the cell lines have been selected, the cells need to be cultured so that there are enough cells on the culture surface to perform screening; however, the culture surface should not be too confluent.  Next, the cells are transferred to multi well micro-plates of various well densities (48, 96 and 384). Drugs and compounds, of appropriate concentrations are added to the wells which are left to incubate or kinases to react with the cells. The cell type, plated cell density and the incubation time differ from assay to assay based upon the experiments. Post –Assay processing is performed by labelling cells with different fluorescent tags. The mostly commonly -used fluorescent labels are fluorescent proteins, fluorescent dyes and antibodies. The excitation and emission spectra of the fluorescent tags need to be carefully analyzed with respect to target identification (Haney, 2008).

### 1.2.2 Image Acquisition

After the cells are treated with the targeted drug, they are ready for acquisition of microscopic images. The choice of microscopic hardware is dependent on two types of microscopy systems: confocal and wide field. The selection is based upon microscope resolution, the size of the object, and the information that needs to be collected. For instance, a confocal microscope uses a pin hole to eliminate out of focus light, thereby providing better resolution; by contrast a wide field microscope includes all out of focus light, but has more signal to noise ratio and fast acquisition time. In order to achieve an appropriate measurement of the targeted objects the acquired images from the microscope must not contain out of focus light, a good image resolution, and no over or under exposed light. To achieve this goal, proper selection of the exposure parameters, excitation filter, emission filters and focus parameters of the microscope is essential (Niederlein, Meyenhofer, White, & Bickle, 2009).

### 1.2.3 Segmentation

Segmentation is the third, and mostcrucial process on high throughput image automated screening. The purpose of segmentation is to identify the targeted objects where measurements need to be calculated. There are various predefined algorithms for segmentation that are commonly used to identify targeted substances in High content screening analysis. The most popular segmentation strategies are Ostu's method (N, 1979), Sobel (Sobel et al) and Canny(Canny, 1986) . Combination of all these segmentation methods also results into obtaining satisfactory results. The output of the segmentation process is the masks images which are binary images that locate the objects of interest for quantifiable measurement (Niederlein et al., 2009)(Oberholzer, Ostreicher, Christen, & Brühlmann, 1996).

### 1.2.4 Feature Extraction

Once the segmentation process has been successful, the output mask images locating the identified objects are used to extract quantifiable features. Different statistical approaches are used to extract various features, such as, texture, morphology, and intensity. These features are extracted from single or multiple fluorescence channels from the same field of view. In high content analysis, huge volumes of data are generated from feature extraction, often in the terabytes (Wollman & Stuurman, 2007) ranges, and not all these data are informative or useful. To get a subset of useful data feature selection methods are used. Formal algorithms, such as Stepwise Discriminate

Analysis and Principal Component Analysis are used to decide the minimum features needed for the proper classification (Kheirkhah & Haghipour, 2010).

### 1.2.5 Machine Learning

After selection, the subsets of feature extraction, machine learning can be used to classify the data or to identify the similarity and dissimilarity on biological phenotypes with the desired phenotypes. For instance, to find out how similar or different are the extracted features with the desired changes of the labelled sample. Three major classifiers are used: supervised, semi supervised and unsupervised. In the supervised classifier, the model is trained with the sample data or control data sets where the labels of the patters are known, and then the classifier is tested with the extracted features (Yang, Beyenal, Harkin, & Lewandowski, 2000). Common supervised learning models are neural network, support vector machines, and KNN (Wong, 2006). While in unsupervised learning, the learning is performed with unlabeled data set, the computer divides the input cells in different categories. Lastly, the semi-supervised models uses both trained and untrained data.

## 1.3 Project Goals

- To design parallel computation software to reduce the processing time of HCS data and extracting Threshold Adjacency Statistics feature from it.
- To compare the results of low level programming language, such as C, with scripting language, Acapella.

- To investigate the morphology and texture features of MCF-7 cell lines treated with staurosporine kinase inhibitor.
- To find an suitable dye that stains MCF-7 cell lines treated with staurosporine kinase inhibitor.

## 1.4 Motivation

Based on cancer statistics in year 2012, 88,800 Canadian women and 97600 men will be diagnosed with cancer each year. On an average day,  500 Canadians will be diagnosed with cancer and 200 will die of cancer every day(Canada, 2012).    Cancer can be defined in medical terms as a disease of abnormal cell proliferation, where these cells are capable invading other tissue through the blood and lymph systems.   There are over 100 different types of cancer, where the naming convention refers to the organ where the cancer originated.  For instance, cancer that origins in the lungs are referred as lung cancer. Cells are continuously being signalled to proliferate, differentiate or die.  However, in cancer cells, a protein that dictates the signal is disrupted due to a gene mutation.  This allows the cells to proliferate autonomously and spread, causing a tumour.  These signal transducer proteins are called kinases; kinases transduce signals in a cascade pathway from the outer membrane of a cell to the nucleus by phosphorylation.  Phosphorylation is a process where  a kinases add phosphate to an amino acid chain of a protein which changes the characteristics of the phosphorylated protein(Faivre, Djelloul, & Raymond, 2006).  The hydroxyl groups (-OH) of serine, threonine, or tyrosine,  amino acid side chains are the most common target to bond a phosphate molecule for phosphorylation

(Secko, 2011) . To inhibit this pathway, researchers have developed kinase inhibitors as a cancer therapeutic drug. Kinase inhibitors inhibit the signals of different families of kinases by targeting a specific kinase in a group, thus interrupting phosphorylation. Unlike conventional therapy, such as chemotherapy, which fails to discriminate between normal cells and tumor cells, kinase inhibitors are more target specific directed towards cancer-specific molecules. These target specific therapies are more therapeutic and provides less toxicity than chemotherapy. Nevertheless, specificity of the kinase inhibitors, targeting a specific protein kinase, is still challenging and their inhibition selectivity is currently under research (Gasparri, Sola, Bandiera, Moll, & Galvani, 2008) (Karaman et al., 2008). Karaman et al have tested the activities of 38 kinase inhibitors against 287 kinases. They have worked on the affinity factor of these kinase inhibitors and found kinase inhibitors off targeting to unrelated kinases. Despite their limitations, they are considered one of the most promising target based therapeutic treatment for cancer due to their specificity. Understanding the specificity of kinases require complex screening with advance microscopic technologies and image analysis modalities(Fabbro et al., 2002). These screening produces huge amount of data and needs fast processing, thus can only be achieved by high content screening. It is our belief that HCS would provide researchers and biologist with the methodologies and technologies to uncover the mysteries of these target specific drugs. Considering this as our incentive, the motivation of the project is using the kinase inhibitor as our test bed for the experiment of applying a solution to accelerate the HCS processes through the development of fast and advanced parallel textural feature extraction software. The increased speed of analysis will therefore, decrease the time of experiment and increase the number of screening, making kinase inhibitor treatment more viable option for cancer treatment.

## 1.5 Major Contributions of the Thesis

- A new parallel image processing technique to extract textural feature was applied on HCS data. This has been proven to be the most efficient technique by previous researchers. Hamilton et al. (2007) have computed threshold adjacency statistics features to distinguish sub-cellular localization of cells more efficiently and accurately than other image statistics computations. TAS has also been widely used in the biomedical field primarily for protein subcellular localization. By understanding the behavior of all expressed protein will simulate cell behavior and therapeutic efficiency. Even though the algorithm has proven to be faster and more efficient than other image statistics algorithms, the algorithm's performance was never tested on parallel model. To our knowledge, this is the first attempt at applying a parallel image processing technique to TAS.

- A parallel program was developed for feature extraction in C language for HCS data. Implementation for the parallel model for computation and I/O intensive HCS data is not a trivial task, due to its inherent complexity and error-prone nature. A minor error in the code can lead to a race condition scenario or deadlock (Messerli, 1998). The difficulty of developing parallel models is one of the major factors preventing commercial software for developing their own parallel image analysis software, which currently run primarily on single processor systems.  In summary, this project develops, for the first time, a parallel C program for TAS feature algorithm and apply it HCS data. The performance is then measured and compared to the commercial offerings.

## 1.6 Thesis Organization

In this chapter the motivation for the project as well as an introduction to the facets of the project is provided.  In Chapter 2 the background information on texture features and subcellular localization features are discussed.  In Chapter 3 the problems and challenges involved in HCS software are mentioned.  In Chapter 4, various solutions, their advantages and disadvantages, with an emphasis on parallel computation as a solution are introduced.  Then the experimental setup and software implementations are explained in details.  The results of this research, conclusions and future work are presented in Chapter 5.

# 2 Subcellular Localization Features

After explaining the concept of HCS, this chapter elaborates on background information in image analysis, texture features and existing texture feature extraction algorithms in the context of subcellular localization features. An efficient feature extraction algorithm, Threshold Adjacency Statistics (TAS), is also introduced, used primarily for extracting meaningful numerical descriptors of subcellular localization of proteins. An example and statistical interpretation of TAS follows, and compares the performance of TAS with existing feature extraction algorithms, such as, Haralick, Zernike Moment, and Local Binary Pattern.

## 2.1 Image Analysis

According to Gonzalez, an image can be defined as f (x, y), where x and y denotes the spatial plane of the image and the amplitude of any coordinates in the plane represents the intensity. Digital images are images processed by computers with finite elements, each representing a particular value and location of the image, referred as pixel. The image is represented as 2D array, and each element of the array is a pixel corresponding to a particular value and location. Base on the range of possible values that a pixel can hold, images can be divided into three types. A **binary image** can only hold two possible values (0 and 1) in each pixel. This type is often obtained by thresholding a greyscale image: any values above a threshold are presented as a 1, and values below the threshold are presented as 0. A **greyscale image** typically has a bit depth is 8 bits, providing a range of possible intensity values from 0 to 255. Finally, a **color image** is represented by 24 bits for each pixel, with the brightness and color information combined (Jain, 1998). Image analysis can be defined as an operation that is performed on an image to extract meaningful information. If these operations are applied in digital images, it is referred to as digital image processing. The processing of an image can be simple, or as complex as facial recognition. In the biomedical field, image analysis is applied to quantify phenotype properties of cells, such as the aspect of shape, intensity, co-localization, texture etc. Among all of the feature extraction algorithms applied in cell imaging, particularly focusing on subcellular localization features, texture is the mostly commonly used numerical descriptor (Wong, 2006).

## 2.2 Texture

Texture can be defined as the variation of intensities across an image, or a variance of pixel values from one pixel to another pixel within an image or local portion of an image. Texture can also be defined as a spatial distribution of greyscale pixels, and their relationship with neighbours (Haralick, R.M.K. Shanmuga, 1973). Texture analysis provides the most significant information in the biological field. This analysis can be done by extracting information from the interested region and differentiating it with another texture group (Haney, 2008). For instance, Haralick texture features provide the information about different types of textural measurement on contrast, uniformity and complexity across the region of interest (Haralick, R.M.K. Shanmuga, 1973). There are various textural feature extraction algorithms that have been developed, but their intensive computation is still a challenge for investigators. However, in the context of subcellular localization features, there are only few algorithms that have been tested so far. Among them, the most common are Haralick texture, Zerenike Moments, Local Binary Pattern and Threshold Adjacency Statistics.

## 2.3 Feature Extraction Algorithm for Subcellular Localization

Subcellular localization can be defined as the localization of molecular compounds or proteins in a specific compartment within a cell. The locations of the probe and the protein are correlated, and can be used as a tool in understanding the function of proteins or the molecular probe. Understanding the subcellular localization of protein provides information on the biological activity of the compound that is being localized, and the relationship between the biological compound and the cellular compartment in which it is localized (Liu, 2012). Computational methods for predicting this localization pattern are very important in understanding the biological activity of the cell at an organelle level. Despite their importance, the progress on developing computational methods to extract subcellular feature is still limited (Gao et al., 2009).

However, Murphy et al. have designed a numerical subcellular location feature set to analyze the subcellular distribution of proteins (Murphy, Velliste, & Porreca, 2003). These features include the Haralick texture feature, Zerenike Moment feature, Convex hull and derived feature from morphology. Detail algorithm of these features is described below:

The **Haralick texture feature** is considered one of the most important feature analysis methods, widely used in the biomedical field, as well as in the processing of radar signals and control systems. In the computation of Haralick texture features, four grey level co-occurrence matrices (GLCM) are created. The GLCM is a tabulation of how often different combinations of pixel brightness values (grey levels) occur in an image. The GLCM description of texture considers the relation between two pixels at a time, called the reference and the neighbour pixel (Roumi, 2009). The Haralick texture method requires producing matrices for each angle and for each offset. This produces a total of four for each angular direction (horizontal, vertical, left and right diagonal); and a total of four for offset zero and one. These matrices are very large; with their imensions dependent on the depth of the pixel intensity. Thirteen texture features are calculated from these matrices, following the five steps shown on the flow diagram (Figure 2-1) below, which measure the homogeneity, contrast, complexity, etc. (Haralick, R.M.K. Shanmuga, 1973).

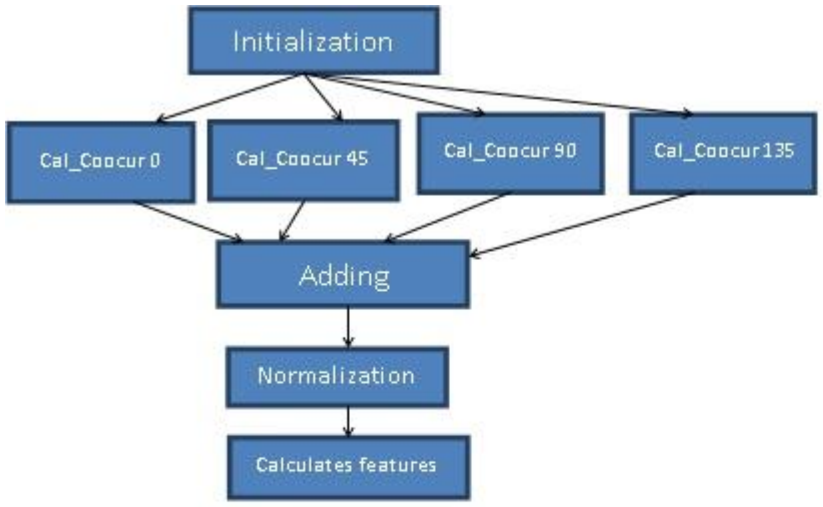


**Figure 2-1: Five steps for calculating the Haralick textural feature matrices.**

**Zernike moment** is another effective statistical descriptor to distinguish subcellular localizations (Hamilton, Wang, Kerr, & Teasdale, 2009). Zernike moment calculates 49 texture features, providing information on rotation and translation invariance.  Zernike moment is calculated by first calculating the center of the mass of each cell, than subtracting the value of each pixel with the center of the mass and dividing the result by the user-specified cell radius R.  Then, to find the ratio of similarities between grey level pixel distributions, the correlation between the transformed image and Zernike polynomial is performed.  Only the amplitudes of pixels within the unit circle of the normalized image are used(Lu, Lu, Liu, & Yang, 2010)(Liu, 2012).  One of the disadvantages of Zernike features is that it is can only be applied to single cell images. As a result, each cell needs to be cropped prior to processing.  Consequently, it requires a very long pre-processing time. For instance, in an experiment by Hamilton et al., pre-processing of Zernike features of 503 images, cropped to 1420 single cells, took 4 minutes and 16 seconds. The total processing time was 17 minutes and 22 seconds to extract the full feature extraction algorithm.  As well, when Zernike moment was tested as a classifier compared with Haralick and TAS, it provided the lowest accuracy of 68.2%, compared to others 86% and 83.3%, respectively.

**Morphology** is commonly used to define the shape of the image: the boundaries, skeleton and convex hull (Mcandrew, 2004). It also provides information related to the shape of the cell, such as perimeter, area and ratio. From a mathematical perspective, morphology is presented in set theory, where two objects in an image can be identified as two sets, set A and set B. There are two main morphological operators: erosion and dilation. Erosion subtracts the value of a pixel from the border of the image, while dilation adds the value of a pixel on the image. Other operators, such as fill and connected, open and closed, boundary and skeleton, are based on the erosion and dilation operator (Gonzalez, 2008). Understanding the shape of the cell or organelles provides the user with significant biological information. For instance, staining MCF-7 cells with Draq5 will provide information of the nucleus, specifically whether the cell is going through mitosis or apoptosis. A condensed nucleus is indicative that the cell is going through apoptosis, with chromatin condensation (Mooney, Al-Sakkaf, Brown, & Dobson, 2002), whereas a round cell shape indicates that the cell is poorly attached (Haney, 2008), etc. In the field of subcellular localization, tagged proteins can be useful in segmentation of different subpopulations of cells growing together, such as neurons.

## 2.4 An Efficient Feature for Subcellular Localization: Threshold Adjacency Statistics (TAS)

Proteins play a major role in cell function, and the location of a protein in a cell can provide vital information for understanding the behaviour of the cell. This is why subcellular location of proteins in cells has become one of the most important studies in biological science (Kheirkhah & Haghipour, 2010). Traditionally, labour-intensive manual visual inspection was the only approach available for identifying the protein's location (Nanni & Lumini, 2008). However, due to the development of fluorescence imaging and image analysis algorithms used in HCS, identifying the subcellular locations of proteins has become easier more manageable task. Efficient statistical modeling techniques are applied on fluorescence-labelled cellular images to extract features from the spatial distribution of proteins. Despite their efficiency, these feature extraction techniques are very complex, and require long computational times with currently available hardware technology. As a result, simpler and faster algorithms are needed in order to extract morphological and textural features of proteins in cells.

In 2007, Hamilton et al. introduced a fast and simple texture feature extraction algorithm for this purpose (Hamilton, Pantelic, Hanson, & Teasdale, 2007a). The algorithm thresholds the image, then counts the number of pixels whose intensities are above the threshold pixel value of "given number of above threshold pixels adjacent" (Hamilton, Pantelic, Hanson, & Teasdale, 2007b). The algorithm was tested with two types of images labelled with endogenously expressed and transfected proteins, providing 98.2% and 93.2% classification accuracy, respectively. It has also outperformed other commonly used texture feature algorithms, such as Haralick and Zernike moment in terms of accuracy. (Hamilton et al., 2007b).

### *2.4.1 Threshold Adjacency Statistics (TAS) Algorithm*

Unlike other algorithms, the computation of TAS is simple. The image is made binary with certain threshold values, and this is performed to distinguish the dissimilarity in the threshold image, which is not possible with the naked eye. Then from the threshold image, for each white pixel, the number of neighbouring white pixels is counted. Therefore, the first TAS value would be total number of white pixels with no neighbour, the second TAS value would be the total number of white pixels with one neighbour, and this succession will continue until nine TAS values are calculated. Finally, these values are normalized by dividing all of the computed values with the total number of white pixels in the image. The following is an example that demonstrates the mathematical calculation of TAS.

**Example 1:** The example demonstrates the computation of Threshold Adjacency Statistics (TAS) by using a 3X3 Matrix:

Image matrix (3X3):

| | | |
|---|---|---|
| 8 | 7 | 11 |
| 3 | 6 | 9 |
| 6 | 4 | 0 |

***Step 1:*** First the mean of the matrix is calculated. The mean of the above matrix is 54/9=6.

***Step 2:*** The interested region is identified. In this example, the region is (mean-3, mean +3); which give us a range between 3 and 9 So now, by thresholding the values between these numbers, the following matrix is obtained:

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

***Step 3:*** The total number of white pixels is found to be 5.

***Step 4:*** All of the white pixels are taken and the white pixels in their corresponding nine neighbours are counted. All the values that are zero are X, because TAS algorithm is only interested in white pixels. For instance, for Row 1 and Col 1, there are two white pixels surrounding it, so the number of white neighbouring pixel is 2. The resulting matrix of adjacent neighbourhood is as follows:

| | | |
|---|---|---|
| 2 | 2 | X |
| X | 4 | X |
| 2 | 2 | X |

***Step 5:*** The TAS is calculated by normalizing it with total white, according to the algorithm.

TAS 0= Number of pixels with zero white neighbours = 0

TAS 1= Number of pixels with one white neighbour=4/5=.8

TAS 2= Number of pixels with two white neighbours =0

TAS 3= Number of pixels with three white neighbours =1/5=.2

TAS 4= Number of pixels with four white neighbours =0

TAS 5= Number of pixels with five white neighbours =0

TAS 6= Number of pixels with six white neighbours =0

TAS 7= Number of pixels with seven white neighbours =0

TAS 8= Number of pixels with eight white neighbours = 0

### *2.4.2 Application of Threshold Adjacency Statistics*

TAS is commonly used for analyzing the spatial distribution patterns of protein in a cell to understand the biological activities of the cell. Researchers are currently evaluating the performance of the algorithm and comparing its performance with other algorithms, by testing with different machine-learning tools. Few are optimizing the algorithm to increase its efficiency. Two of the major developments are provided below: Fatema *et al.* have used a different threshold adjacency statistics algorithm, a modified version, to extract the sub-cellular protein location features. The algorithm is similar to Hamilton et al but instead of computing the threshold adjacency statistics in one layer around the targeted pixel, it calculated three layers of pixel in each white pixel. The algorithm was tested with same set of images, test sets of images by Hamilton on similar model of Support Vector machine, provided an accuracy of 97.06%, and outperforming the results of Haralick and Zernike moments (Kheirkhah & Haghipour, 2010).

Fatema *et al.* have used a modified TAS algorithm to extract the subcellular protein location features. The algorithm is similar to Hamilton et al, but instead of computing the threshold adjacency statistics in one layer around the targeted pixel, it is calculated by three layers of pixels surrounding each white pixel. The algorithm was tested with the same set of images as Hamilton on similar model of Support Vector machine (SVM), and obtained an accuracy of 97.06%, outperforming the results of Haralick and Zernike moments (Kheirkhah & Haghipour, 2010).

Lorins, Nanni et al have used a different approach to improve the accuracy by combining different features and classifiers on the 2D-Hela dataset. The research used two different classifiers to test the accuracy: the SVM and the neural network. It also used features of Local Binary Patterns, TAS and Haralick. The results show the neural network performs better than SVM and the combination of Haralick and TAS obtains better accuracy, 98.2% in endogenous dataset and 93.2% in transfixed dataset (Nanni & Lumini, 2008).

### *2.4.3 Comparison between TAS and other Texture Feature Algorithms*

Texture feature extraction methods are the most widely used algorithm in the medical field. However most of these texture feature algorithms are complex, computationally intensive and time-consuming. For instance, Haralick texture feature (Haralick, R.M.K. Shanmuga, 1973), the most commonly used feature extraction algorithm, requires creating four grey level co-occurrence matrices in four different angles for each offset, which requires very long processing times as well as large storage considerations. In contrast, TAS does not have any computational overhead besides the size of the threshold matrix that it generates. When Hamilton et al. Processed 503 images with TAS, Haralick and Zernike feature algorithms, it took 12 and 18 minutes for Haralick and Zernike to extract subcellular features, respectively, while TAS took 60 seconds for 27 textural features (Hamilton et al., 2007a).

Another major drawback of other textural feature algorithms is that they require pre-processing prior to implementation. Pre-processing is a two-fold problem: requiring additional time, as well as providing reduced accuracy (Nanni & Lumini, 2008). For instance, Zernike feature extraction requires pre-processing time for single cell cropping, and automated cell selection for cropping may provide variable results by failing to locate all cells. To illustrate this problem, in the same experiment mentioned before by Hamilton et al., it took 5 minutes of additional time to crop 503 images into 1420 single cell images. In contrast, TAS only requires pre-processing for making binary images by thresholding: a 30 percent reduction in processing time (Hamilton et al., 2009).

# 3 PROBLEM DESCRIPTION and Parallel Processing

After exploring the background of HCS, feature extraction algorithms, threshold adjacency statistics and parallel processing, this chapter will present the current challenges and weaknesses of HCS software in general. The first section illustrates the deficiencies of customizability and multiprocessing in current HCS software, as well as the shortcomings of high-level scripting language compared to low-level C languages in image analysis. The second section follows by analyzing the limitations of contemporary three image analysis software packages: cellprofiler, ImageJ and Acapella. Finally, in the next section (2.3), it introduces parallel processing and existing parallel models.

## 3.1 Challenges of High Content Screening Analysis Software

Even though optical imaging modalities have advanced, the hardware and the software used for image analysis are still in a comparably primitive stage (Eliceiri et al., 2012). Due to the extensive development within this field, vast amounts of multiparametric data are generated that cannot be analysed by the software and hardware at a fitting pace. In other words, the algorithm of the image analysis software is still not optimized for the needs of the HCS system. The feature extraction algorithms, which are developed for different fields and for unique applications, now need to be adjusted and customized according the requirements of HCS data processing. Current algorithms, such as Haralick and Zerenike moments, are no longer suitable for ongoing HCS technology, which needs further tuning and optimization to provide faster and more accurate results. Alternatively, Threshold Adjacency Statistics (TAS) is a new approach that could provide a quicker algorithm that would provide the same magnitude of accuracy with faster performance. On the other hand, along with a suitable software algorithm, the design and selection of appropriate hardware, compatible with assay based experiment, is also vital. Due to their limited processing power, built-in chips lack flexibility, and biologists are included towards other customizable processing hardware, such as, multiprocessing, distributed systems and FPGA. The advantage of these hardware options is that they are programmable and expandable according to the need for screening. With proper design and implementation these hardware options can provide satisfactory performance, both in speed and accuracy. Despite the flexibility of these architectures, the major challenge is to develop an efficient and optimized code that could fulfill the requirement of HCS analysis, without compromising speed or accuracy.

Another major bottleneck for HCS is the lack of flexibility and customizability of the image analysis software. Most of the image-processing software that is bundled with the microscopes is mainly developed for drug discovery intended for the pharmaceutical companies (Niederlein et al., 2009). These software modules only support functions that are needed for drug testing purposes; therefore, they lack flexibility for use in customized screening. They are also very expensive, and the modules only support image processing of mammalian cells and cellular features (Carpenter et al., 2006). Nevertheless, few have used their own customized scripts and alternative scripting languages to overcome the limitations of the commercial software. Despite the offered scripting language's moderate flexibility compared to the commercial software, it is still dependent on the supported library features, and it is slow due to the translation to machine code. Furthermore, the major drawback of both of these software options, commercial and scripting, is that their algorithms are proprietary software, which lack flexibility of any modification and cannot be customized for various complex screening experiments. The software's source code is hidden, so the algorithms cannot be modified according to the experiments. This prevents the end users from being able to write codes which are perfectly suited for their experiments.

An additional downside of image analysis software developed in a scripting language, such as Acapella, is that they make use of high-level languages, which are further away from the hardware, reducing speed and performance. High-level languages make it easier for the programmer to code and understand the program thus requiring lesser programming skills. However, this convenience to the end user is provided by compromising the flexibility of the algorithm design at the hardware level. For instance, low-level languages, like machine languages and assembly languages, require detailed knowledge of the hardware, which provides more design flexibility to the programmer due to their direct interaction with hardware. On the contrary, high-level languages are syntax-specific, requiring less programming skill: only the knowledge of the supported syntaxes (Ram, 2005). Therefore, they are easier to program but lack design flexibility. Since low-level languages have more control over the hardware, they are faster and more resource-efficient. High-level languages are further away from the hardware, and they are dependent on an interpreter or compiler to translate the code into low-level code. The translation process from higher level to machine level makes the scripting language slower. For instance, codes in the Acapella scripting language use an interpreter, a tool that translates each line of code to machine code as it is executed during run time, to decode scripting syntax to hardware level and an interpreter would require more time to translate to a hardware platform than an intermediate-level programming language, such as C, C++ or Java. Due to its control over the hardware, intermediate level languages can perform better with finite resources by using resource optimization techniques in the code. Programmers can design a task with limited resources by optimizing the memory usage, processor speed and storage.

Another shortcoming of the existing software is that they lack the capability of multiprocessing. The majority of software is built for a serial processing environment. As a result, they typically fail to use a multiprocessing environment efficiently. Serial processing software runs a program in a single execution path, on a first-come-first-serve basis. As a result, they fail to fully exploit the multi-core environment. Furthermore, the load sharing of jobs in serial processing software are not evenly distributed to the entire core. Therefore, optimized hardware efficiency is not achievable. Making the situation worse, due to their lack of accessibility in the code, end users are also incapable of modifying the code in order to optimize it for multiprocessing technology, like second generation languages C, Java, etc. Most of the HCS feature extraction algorithms are repeatable operations, meaning the same code is run on different images multiple times. Therefore, the goal of implementing code efficiently for parallel environment is of significant interest for the biologist. In summary, due to their proprietary nature, lack of flexibility, multiprocessing incapability and further limitations, there is a clear need for software that would be more customizable for screening needs, with faster processing times.

## 3.2 Existing High Content Analysis Software and Their Limitations

There are two types of existing software in the image analysis arena: 1) proprietary software 2) open-source software (Niederlein et al., 2009). The proprietary software can be further divided as the one that comes bundled with the microscope, and separate image analysis software that can be integrated with microscopy images. Microscope packaged software is limited to the processing features and operations supported by the microscope, compared to customary proprietary image analysis software. Despite their easy-to-use features and minimal programming skill requirements, proprietary software packages are extremely expensive and are limited to built-in features (Lamprecht, Sabatini, & Carpenter, 2007). However, open-source software is not typically restricted with any license, and is less expensive overall. Unlike proprietary software, their software packages are flexible and the coding analysis is not hidden for the users. However, they require more extensive knowledge of programming to adapt, depending on the software macros and custom algorithms. For instance, ImageJ and Cell profiler are commonly used open source software, but it requires skill on Java and Matlab programming language.

The project uses Acapella scripting language to compare the performance against the feature extraction program developed in programming language C. Acapella is one of the bundled software packages that comes with the Opera image acquisition microscope from Perkin Elmer. The input image format in Acapella is usuallyFLEX, converted to TIFF during processing. The Acapella program runs on scripting languages. To process images, it uses user-friendly drag-and-drop modules, as well as an alternative optional text-based editor. Nevertheless, the scripting language is user-friendly and does not require any prior background on programming. Users can select different algorithms, such as nuclei, spot and cytoplasm detection, for segmentation purposes (Elmer, 2008). It also has a partially-open architecture, providing flexibility in writing algorithms based on advanced assays *(http://www.perkinelmer.com/; access September 23, 2012).* However, Acapella uses interpreter for translation to machine code, making it very slow. Additionally, the scripting environment does not support any multiprocessing features. The syntax is primarily dependent on pre-defined library functions, which provides less flexibility to customize code. Due to the limitations on these pre-defined functions or API, Acapella generates lots of images that takes lots of storage space and time. For instance, to generate nine TAS values in Acapella generates 25 mask images whereas the C code only generates one mask image.

## 3.3 Parallel Processing

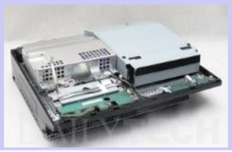
A huge amount of data is required in order to process HCS data, which is a significant problem for a conventional microprocessor. To execute 2000 cells for TAS factors would require almost 7 hour of processing time. Even an Intel processor with a quad core of 4GHz will require high speed memory access to process the data. L1 and L2 cache are also on the scale of kilobytes, which are not enough to hold one single image. Memory is still slow and the greatest disadvantage is that their bandwidth is limited to one word read/write cycles. The number of transistors and clock speeds of microprocessors will likely continue to increase exponentially according to Moore’s Law, however the memory access time will increase linearly (Greco, 2005). As a result there will be always an ever-widening gap between these two compatible technologies. Hence, in order to improve efficiency and improve speedup on image processing, parallel algorithms need to be developed so that it can run independent tasks in parallel on multi-core or multi-processor systems. Parallel algorithms should also take into consideration data dependency, processor-to-processor communication overhead, and I/O and CPU computation jobs. Parallel computation can be defined as “simultaneous use of multiple computer resources to solve independent tasks concurrently and efficiently.”(Blaise Barney, 2012)(Grama, Aananth, Gupta, Anshul, Karypis, 2003)

### 3.3.1 Existing Image Processing Model and Their Limitations

Table 3-1 represents the current available hardware technologies for parallel processing with their limitations. The major limitation of these technologies is that they require specific hardware or special skill in programming. For instance, FPGA requires HDL programming language skills, and NVidia requires CUDA and Cell Broadband Cell SDK 3.1. There are also other specific limitations. FPGA often needs an extra memory block or symmetric images for better performance. GPU performs less efficiently on biomedical images, and CBE performance is measured based on throughput instead of the simplicity of the algorithm (Shahbahrami, Pham, & Bertels, 2011). As a result, the project was implemented on core microarchitecture because they are simpler, more available and can be integrated with existing microscopic system without any additional cost.

**Table 3-1: The table represents the current multi core hardware technologies and their limitations.**

*Images Ref: website Altera, NvVdia Sony Playstation,intel (Reteieved September 3, 2012)*

| Architecture | Hardware Platform | Limitation and Language Platform |
|---|---|---|
| FPGA | Altera Stratix IV GX FPGA | ❖ HDL<br>❖ External memory block.<br>❖ Symmetric images. |
| Graphics Processing Unit (GPU) | NVidia GeForce 600 Series | ❖ CUDA<br>❖ For Biomedical imaging performance less than FPGA.<br>❖ OpenCL new approach. |
| Cell Broadband Engine (CBE) | Sony PlayStation 3  | ❖ Cell SDK 3.1<br>❖ Performance is base on throughout, not algorithm. |
| Core Microarch. | Intel Core | ❖ Adaptable to most compiler and traditional programming tool.<br>❖ Easily Available. |

# 4 Customized Software Solution

After understanding the limitations of existing systems from Chapter 3, this chapter presents the need for parallel processing in high content screening (HCS) as a solution to fast and accurate processing. The chapter examines towards the solution by categorically explaining the design and implementation phases of the parallel feature extraction software of the project. The development phases of the software are divided into four parts: data collection, design of the software, design of the programming code and finally design of parallelization model.

## 4.1 The Need for Parallel Computation in High Content Screening

High content screening analysis (HCS) produces a huge amount of data, often in the size of terabytes (Niederlein et al., 2009), and this requires massive processing power resulting in long analysis time. There are various factors that affect the speed of High Content Screening. For instance, the number of cells on a plate, typically $10^6$, requires a long time to process key features. In a typical cell by cell experiment, each cell is represented in the matrix; therefore, the more cells on a plate will require more matrices to process resulting in more analysis time. To illustrate, in Acapella software, each image is represented by 672x508 matrix whose entries are 16 bit numbers. This requires an outstanding processing time for each typical experiment that uses $10^6$cells on average. Another big factor that influences the processing is the number of features that needs to be calculated, and the complexity of the feature calculations**.** Feature extraction calculations are mostly repeated structures, meaning that the same code or task with different parameters is called multiple times for each image. Therefore, as the number of images increases in the analysis, the computational power increases, and so too does the time of processing. As a result, high content screening with a single-core processor system would take an extensive amount of time, storage and processing power, providing a less efficient method. To understand the properties of HCS data, a set of sample HCS data has been collected from Biophotonics Lab. A probability density function of this data set is provided in Appendix A.

To overcome the limitations of serial computation in high throughput screening, a parallel computation would be the most efficient approach to reduce the time. The advantage of parallel processing is that several computations can be processed in simultaneously. In parallel computation approach, large tasks are divided into discrete independent tasks, so they can be executed in multiple processors concurrently (Blaise Barney, 2012). As mentioned earlier, most of the feature extraction tasks are repeated codes that are executed multiple times on a same image. In a parallel computation environment, these codes can be executed in parallel using multiple processors simultaneously, instead of sequentially in a serial processing system. Thus, it would make the HCS feature extraction software run much faster than before. In addition, in terms of image size, the large number of bytes of images in HCS can be partitioned, and each portioned datasets can be processed in parallel. Furthermore, the problem or the task applied on the images can also be divided into several independent subtasks, where each subtasks can be executed in parallel as well. To summarize, both parallel design approaches, partitioning the data sets or the tasks, would accelerate the image automation process of HCS.

## 4.2 Data Collection

The data collection of this project can be categorized in two parts: 1) Sample Preparation and 2) Image Acquisition

### 4.2.1 Sample Preparation

Figure 4-1 shows the task breakdown of the laboratory experiment. The quality of the assay is very important for image processing: a better assay provides a higher quality image, which reduces the speed of processing (Lu et al., 2010). During this screening assay, a systematic protocol for staining, drug dosing, and image acquisition was maintained for better image quality. Different experimental variable, such as, the number of cells to avoid confluence, correct focus to avoid out-of-focus light, and optimized staining for better quality image were seriously monitored (Haney, 2008).

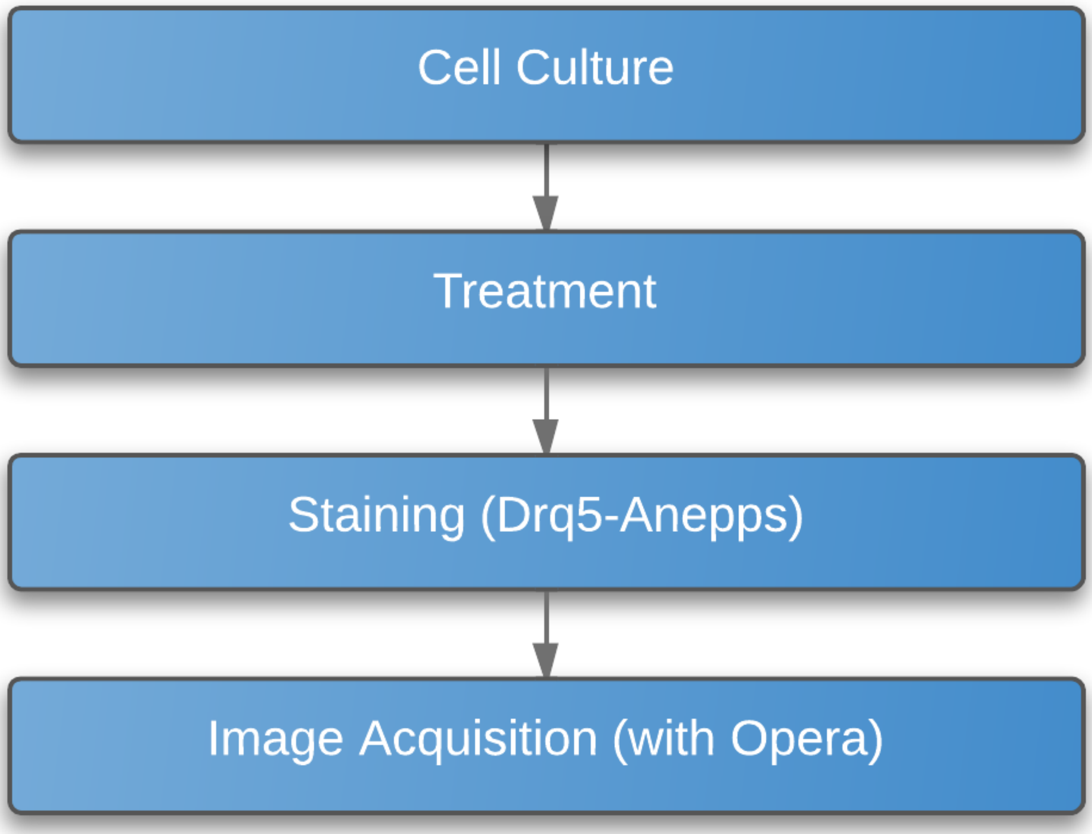


**Figure 4-1: Task breakdown of data collection and image acquisition.**

#### *4.2.1.1 Cell Culture and Slide Preparation*

Our experiment uses the MCF-7 breast cancer cell line. The cell line is collected from liquid nitrogen storage.  To thaw, the cells are placed in water bath (37 -C) for five minutes, then slowly diluted with 10 fold of growth medium (FBS) and placed in incubation for 24 hours at 37C.

After 24 hours of incubation, the culture's cell growth is observed. If the cells reach 80% confluence, the cells are passaged and a small sample is taken for experiment. The subculture process for the MCF-7 cell line begins with washing the cell layer with PBS and then adding trypsin. The solution is then placed in the incubator for 5 minutes to cause the cells to detach from the dish. Finally, the detached cells are suspended with fresh medium, which are then used for welling or incubation. Cell culture is protocol and materials were prepared by ref (Doyle et al., 1995)

#### *4.2.1.2 Treating With Staurosporin*

When the cells are ready they were welled on 384 well plates, each well consisting of 5000-7000 cells. Cell counting is performed with a haemocytometer to ensure a consistent number of cells. To treat the cells, serial dilution is used from highest to lowest dose. The experiments used Staurosporin kinase inhibitor to treat the cultured MCF 7 cell lines starting from high dosage of 1micromolar down to .0004 micromolars. Table 4.1 presents the map of the dosage treatment on the well plate. The first two and last two rows are untreated wells (as controls), and the rest of wells run from highest to lowest concentration. Every new concentration of solution is reduced to half of previous concentration. Staurosporin kinase inhibitor was selected because due to its promiscuity and high affinity with most kinases (Karaman et al., 2008)(Ghoreschi, Laurence, & O'Shea, 2009).

Table 4-1: Well Map of dosage of staurosporine used in the experiment, dose range (1 micromolar to .0004 micromolar)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | untreated | untreated | untreated | untreated |
| 2 | untreated | untreated | untreated | untreated |
| 3 | 1 | 1 | . 015 | .015 |
| 4 | 1 | 1 | . 015 | .015 |
| 5 | .5 | .5 | .0007 | .0007 |
| 6 | .5 | .5 | .0007 | .0007 |
| 7 | .25 | .25 | .0003 | .0003 |
| 8 | .25 | .25 | .0003 | .0003 |
| 9 | .125 | .125 | .00015 | .00015 |
| 10 | .125 | .125 | .00015 | .00015 |
| 11 | .0625 | .0625 | .0019 | .0019 |
| 12 | .0625 | . 0625 | .0019 | .0019 |
| 13 | .03125 | .03125 | .0019 | .00039 |
| 14 | .03125 | .03125 | .00039 | .00039 |
| 15 | untreated | untreated | untreated | untreated |
| 16 | untreated | untreated | untreated | untreated |

#### *4.2.1.3 Staining With Fluorescent Probes*

The major concern with staining is to identify the appropriate dye that would stain the interested organelle or cellular membrane for proper segmentation.  Proper staining protocol is very vital, since it influences the segmentation of interested object--specifically if the analysis is based on an intensity threshold (Ronneberger et al., 2008). Therefore identifying the best candidate dye that stains the cells membrane, which is the experiment's object of interest for subcellular localization, is very challenging.  Several dye were tested, such as, PKH36 red fluorescent, NaO and Di4-Anepps (Kao, Davis, Kim, & Beach, 2001) on MCF-7 cell lines; and the most suitable dye that stains the cellular membrane  was determined to be Di4-Anepps. The excitation spectrum for Di4-Anepps is 450-510nm, and the emission peak is 570nm, after an incubation time of 30 minutes (37C) (Invitrogen, 2012). In addition, Draq5 dye was used for staining the nuclear membranes, used as a reference signal for image segmentation. The protocol for Di4-Anepps was followed from the protocol referenced in Invitrogen (Invitrogen, 2012) , and the Draq5 protocol is referenced from Biostatus (Biostatus, 2012).  Figure 4-2 shows the excitation and emission spectra of Di4-Anepps and Draq5 dye and figure 4-3 shows images of stained cells (figure 4-3):

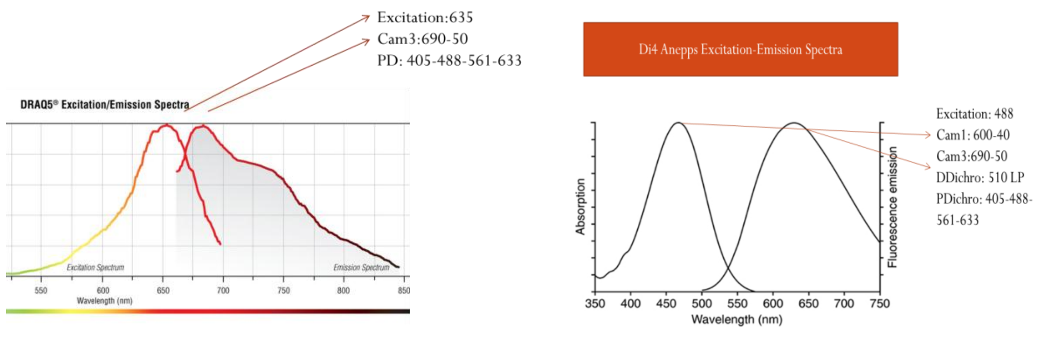


**Figure 4-2: Excitation and emission spectra for Draq5 and Di4-Anepps. Draq5 has excitation peak at 635nm and emission peak at 690nm; and Di4-Anepps has an excitation peak at 448nm and emission peak at 600nm.**

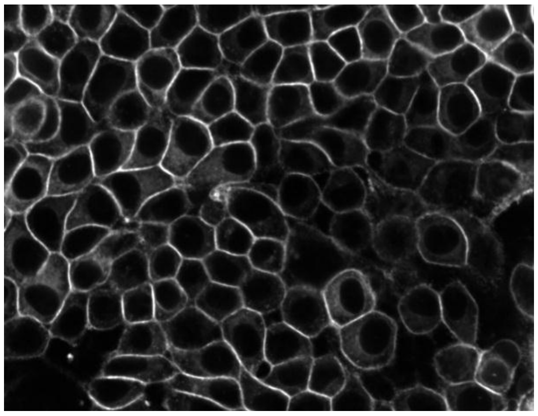
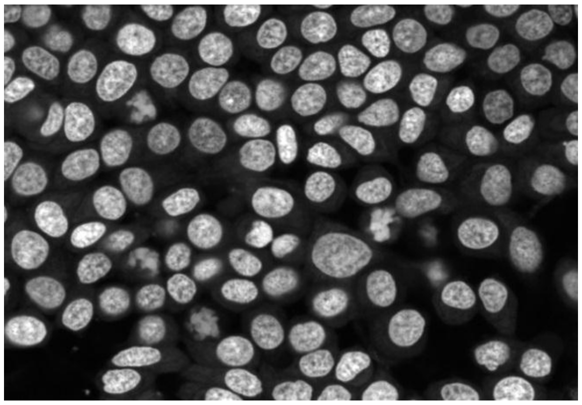


**Figure 4-3 a) Di4-Anepps staining of cellular membrane.**


**4-3 b) Draq5 staining of nucleus.**

## 4.2.2 Image Acquisition

Images are taken by using the Evotec Technologies Opera automated microscope system. Opera is a confocal high-content screening microscope with three laser lines (488,561 and 640 nm) and a UV filter for screening in 96 or 384 well format (Elmer, 2008). The images captured from the microscope are in FLEX format; where one FLEX image stores images of one single well from specified field of views. The experiment has used two cameras to acquire image, three exposures and six fields of view: Exposure 1 on Camera 3 is assigned as Channel 1, as reference channel for the nuclei; and exposure 2 and 3 was taken by camer1 as channel 2 and channel 3, respectively, for the cytoplasm.

Figure 4-4 show the microscopic images acquired from the dose response experiment. Control cells without any treatment are traced with a yellow segmented outline and the cells treated with highest dose of 1 micromolar of staurosporin are traced with a red outline. As seen in the figure, most of the cells are affected after being treated with staurosporin. Figure 4-5 depicts the drug dose curve. It can be seen that at .03 micromolars, 25 percent of the cells are treated. As the dosage increases to 1 micromolar the percentage treated cells increases to 40%.

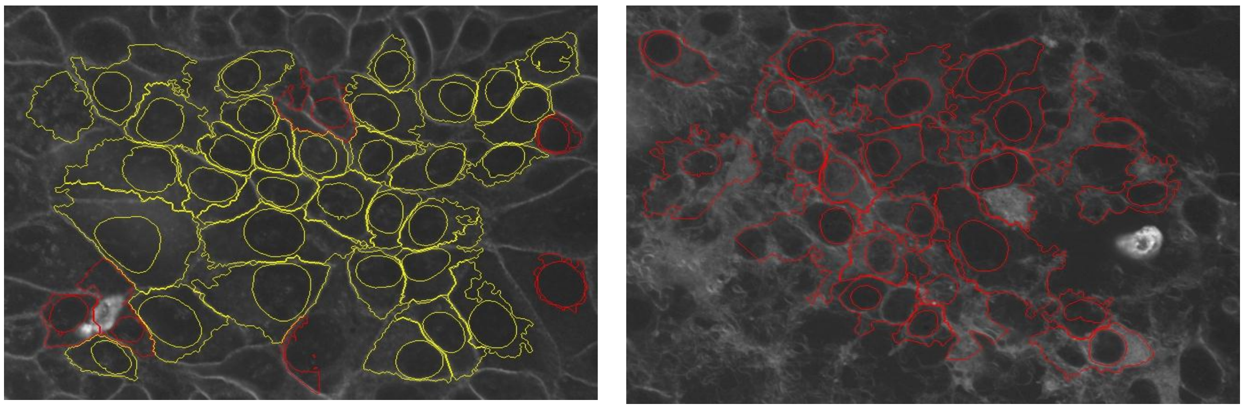


**Figure 4-4: Image of control cells (left) and treated cells with high dosage (right). The yellow segmented stencils represent cells which are not treated and red stencils represent treated cells.**

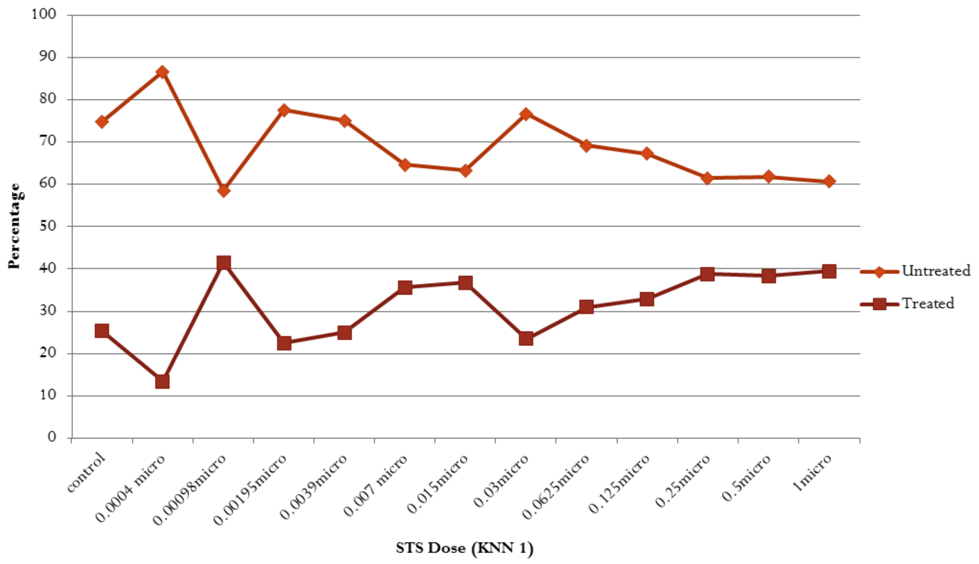


**Figure 4-5: Drug dosage curve of MCF-7 cells with .00004 micromolars to 1 micromolar. Using KNN1 model: at .03 micromolar 75% of cells are untreated and 25% treated; at 1 micromolar 60% are untreated and 40% treated.**

## 4.3 Design of the Software

The design of the software is split into three sections. The first section defines the step by step development process of the software, following the waterfall model. The second phase explains the coding design of the software, with description of all the functions in sequential order. The last phase provides the parallel design of the software.

### 4.3.1 Development Process of the Software

In this project, the goal is to design parallel image processing software to extract texture features from HCS data. The objective is to improve the running time of the program, so the vast amounts of data can be processed in a way that minimizes processing time. In order to boost speed of processing the data, the project was approached in two different ways: algorithmically and by parallel processing. The algorithmic approach requires analyzing the code and reducing the running time of the program. An efficient algorithm with short running time is the required goal of the project. In the second phase, the algorithmic code, written in C, needs to be modularized according to the parallel-processing functionalities. Figure 4-6 depicts the software development process of the parallel image automation system:

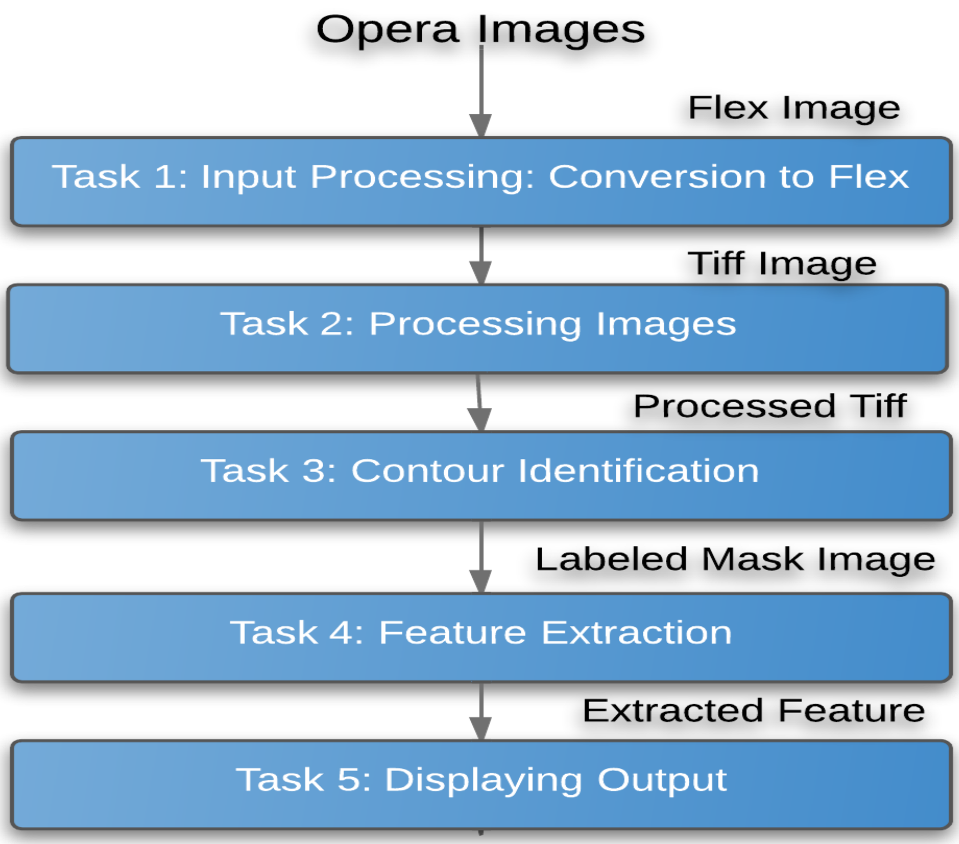

Opera Images
Flex Image
Task 1: Input Processing: Conversion to Flex
Tiff Image
Task 2: Processing Images
Processed Tiff
Task 3: Contour Identification
Labeled Mask Image
Task 4: Feature Extraction
Extracted Feature
Task 5: Displaying Output

**Figure 4-6: Software development process for parallel feature extraction software. The process model outlines the waterfall model- the sequence of tasks to develop the software.**

#### 4.3.1.1 Input Processing: Conversion from FLEX to TIFF

The input is the first step in the development process. According to the requirement, there are two types of input images the program is capable of processing. These two image formats are: the FLEX image and the TIFF image format. Reading the FLEX image format is very complex, since it uses a proprietary microscopy images native to Acapella, and the only way to convert it to TIFF is to use the Bio-Format conversion tool package from LOCI (LOCI, 2012). However, the Bio-Format conversion package is not directly complaint with the C platform, it is for MATLAB and other supported applications. The LOCI conversion tool, for C language, is a package of Java class libraries whose routines are used to convert FLEX images and to extract their metadata. To make it compliant with the C language, it uses the jar2lib program to generate a C++ proxy class for each equivalent Bio-Format Java class (Pepperkok & Ellenberg, 2006). The second image format, TIFF, can be directly read from the input folders. Due to their indirect conversion processes, reading FLEX images requires more time than reading TIFF images. The format of these images is 16 bit; however, the mask images generated by the program are 8 bit images.

#### 4.3.1.2 Preprocessing Images

Preprocessing the process of reconstructing the true intensity values of the fluorophore distribution by removing noise or uneven illumination (Ronneberger et al., 2008). In the experiment, the images captured from the microscope are pre-processed in two stages: removing the border cells and normalizing the intensity values of the cell. Since the cells at the border of the image lack full information of a cell, it is omitted for further processing. Once the cells are identified, the 'remove border' function removes the cells that are on the border of the image. Figure 4-7 shows mask images after removing the border nuclei. The second stage of preprocessing is, applying smoothing techniques (Fotiadis, 2002). It is done by computing the average intensity of each object and distributing it on the full object.

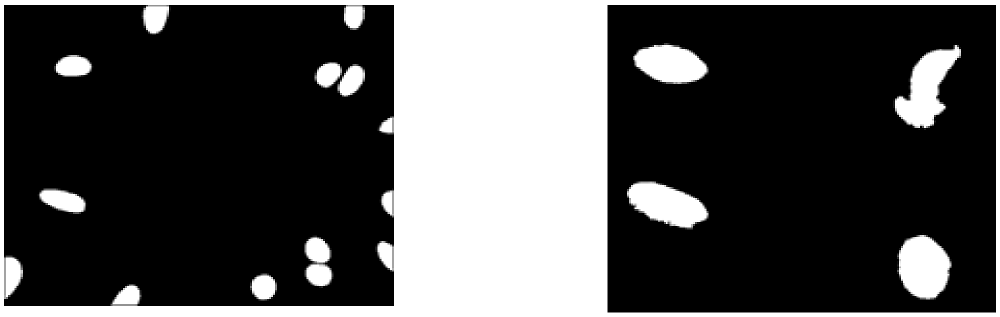


**Figure 4-7: Mask image before preprocessing (left) and mask image after preprocessing (right). The border nucleus is removed from the images on the right.**

#### 4.3.1.3 Contour Identification and Segmentation

Segmentation in fluorescence imaging is mostly based on intensity thresholds, either for edge detection or region selection (Yang et al., 2000). In this experiment, the segmentation process is initiated by first identifying the nucleus and cytoplasm using the watershed algorithm. Each image has a field of view which includes many cells. Two stencils are segmented – there is one stencil for the outer membrane and one for the nucleus. These two stencils are used to identify the whole cell, the nucleus and the cytoplasm, which is the whole cell minus the nucleus. Acapella performs the initial transformation of the data on the whole image and then uses the stencils to calculate the features on a cell by cell basis. Our C code takes these stencils to identify the nucleus, cytoplasm and whole cell, and extract the textural features. The program separates each stencil into a separate mask image, and uses that mask image to extract feature from the image. The code uses a contour identification algorithm to identify each stencil from the mask image generated by Acapella.

#### 4.3.1.4 Feature Extraction

Texture feature, threshold adjacency statistics (TAS), is extracted from the images, which are recognized by the contours. Three TAS values were calculated for each object. These three TAS values correspond to different threshold values applied to the objects. For each threshold, nine TAS values are computed, totalling to 27 for three thresholds of each identified object. Following are the range of threshold:

$[\mu - \Delta, 65535]$

$[\mu+\Delta,,65535]$

$[\mu - \Delta, \mu + \Delta]$

where $\mu$=average intensity, and $\Delta = \mu*\text{range}$

For every threshold, a mask image is created of the corresponding threshold values. This mask image is used to perform the TAS calculation. The TAS is then calculated for each white pixel, and the total number of neighbouring white pixels is counted by looping around its eight neighbours on the matrix. Nine threshold statistics are computed, each representing the white total number of white neighbours around each white pixel. Finally, each threshold statistics is normalized with total number of white pixels.

### 4.3.2 Implementation of C Code

A control flow diagram provides an overview of the sequence of functions (or a flow chart of function calls) that was executed in the code.  Figure 4-8 is the control flow diagram of the software.  This is followed by a description of each function**.**

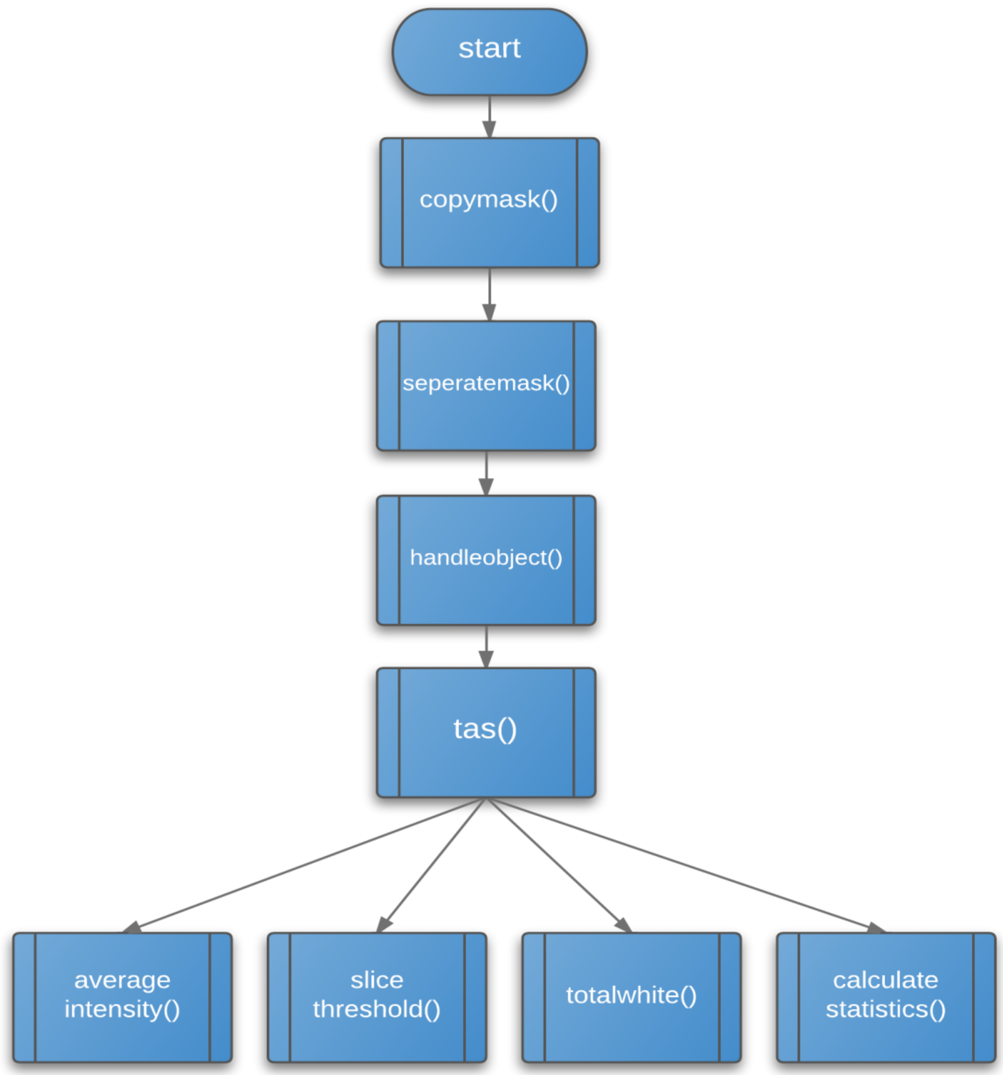


**Figure 4-8: Control flow diagram describing the sequence of function call in the code.**

***copymask ():*** The purpose of this function is to initiate the program execution, initializing the global variables and reading the location of the folder of the mask image folder.  The function loads three mask images: nucleus, cytoplasm and whole cell.  Instead of loading the cell mask, the function derives the cell mask by adding the nucleus and cytoplasm masks together.

***separatemask ():*** The purpose of this function is to separate each contour from the mask with the image provided by the copymask function. Each contour is saved in a separate image.

***handleobject():*** The function takes the separated contour image generated by the separatemask function, then loads the TIFF input image from each channel, and performs a logical 'AND' operation on both of these images. This allows the program to extract the intensity information of each image on the location of the contour. The function calls the TAS function for each channel to compute the threshold adjacency statistics for that located object.

***tas ():*** The function performs the computation of TAS of each object located by the contour from each channel. The computation is made by calling the four following functions, each performing the specified tasks:

***average intensity():*** The function computes the average intensity of the object.

***slice_threshold():*** The function creates a mask image based on the threshold values.

***total_white():*** The function counts the total number of white pixels in the mask image created with the threshold values by the slice_image function.

***calculate_statistics ():*** This function counts the total number of neighbouring white pixels of each white pixel. The function calculates nine TAS values; each corresponds to the total number of white neighbours for each white pixel in the threshold image.

These four functions are called by tas() for calculating the TAS of one threshold value, producing nine TAS values. The functions will need to be invoked three times for the threshold values, generating a total of 27 TAS outputs for each object.

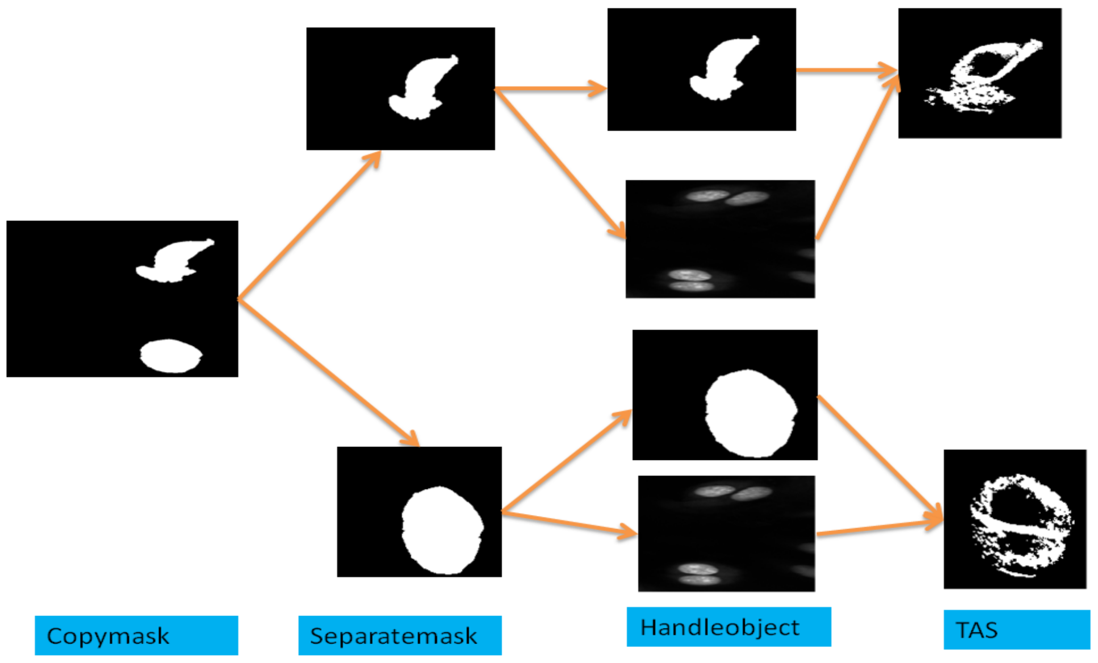


**Figure 4-9: An overview of images generated as each of the functions labeled below is called.**

Figure 4-9 provides an overview of all the objects that are generated as each function is executed. The *copymask* function reads the mask image and calls the separatemask image. The *separatemask* function creates separate images of each object and invokes the *handleobject* function. The *handleobject* function then loads the original grey-scale image and perform an ‘AND’ operation with the single mask image. The resulting image is sent to TAS to perform the threshold adjacency calculation.

### 4.3.3 Design of Parallelization

#### *4.3.3.1 Overview of the Design*

As mentioned earlier, parallel design requires breaking down the task or the computational problem into discrete components such that each task can be executed independently and distributed to multiple processors (Grama, Aananth, Gupta, Anshul, Karypis, 2003). Each task can be categorized as being dependent on other tasks, being independent of other tasks, or requiring intensive computation. After arranging the tasks, hotspots of the tasks are identified. Hotspots are single tasks or a collective task that require intense computation time. These tasks are further analyzed to reduce the time either by using optimized algorithm or by breaking them down into more independent tasks. Since reading and writing requires massive computation time, I/O intensive jobs are also labeled, so they can be more evenly distributed among different processes. The goal of the ideal parallel model would be to parallelize and synchronize tasks that can execute independently, and distribute the load of all processes evenly. The size of these tasks is important in parallel processing and must be optimized carefully (Silberschatz, Galvin, and Gagne, 2004). This is because the computational overhead of parallelizing a small task can actually be detrimental, while large jobs will acquire the processor for long periods of time blocking other jobs from executing, thus decreasing the performance. Following these design principles, the project's parallel model was created, shown in Figure 4-11. The dependent tasks or processes, tasks that are dependent on previous tasks, are presented along the horizontal direction, and are done sequentially following matching-colored arrows. Independent tasks can be executed in parallel, and are listed vertically. Tasks that are independent of each other are shown in the same colour and fall in the same dotted-line region.

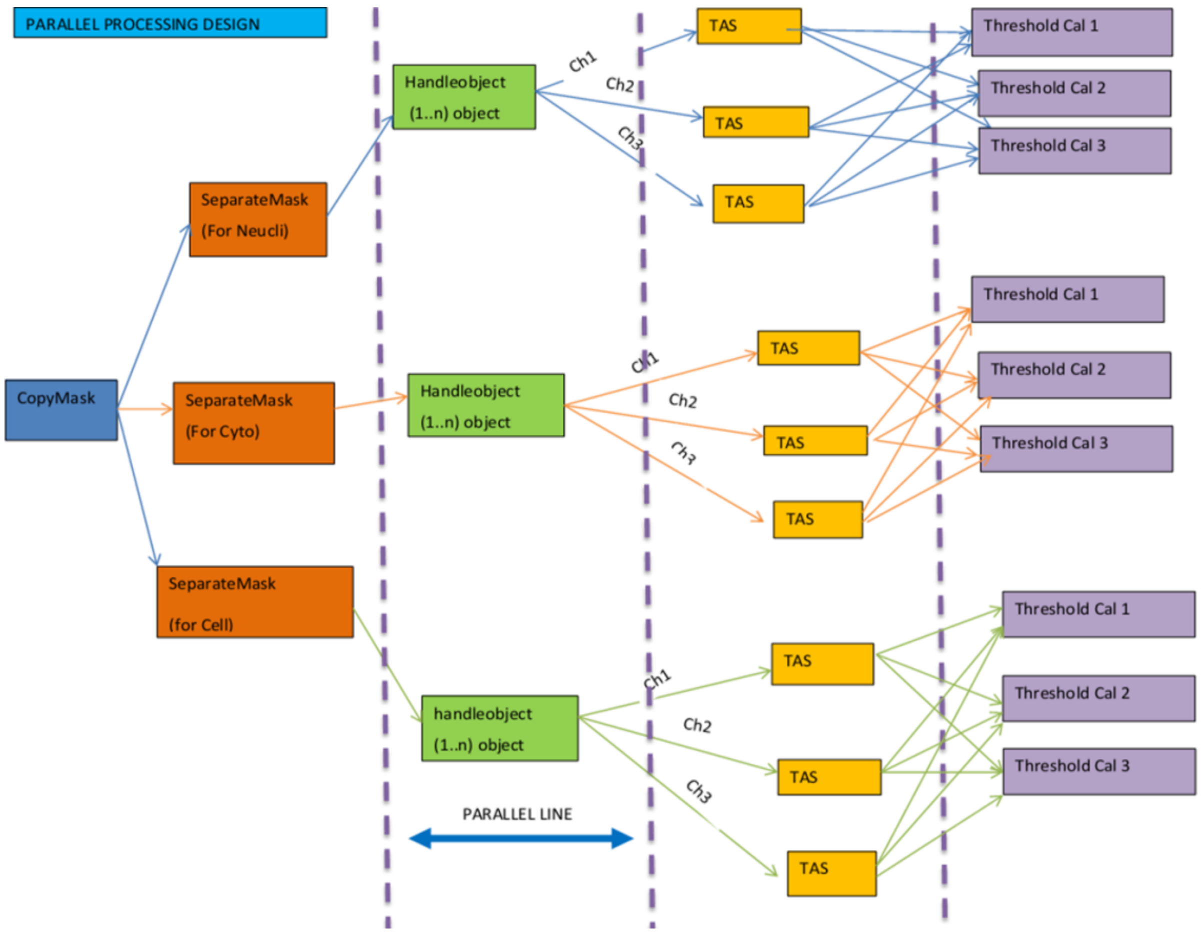


**Figure 4-10: Parallel design of the feature extraction software. All functions present in a parallel line can be executed in parallel.**

### *4.3.3.2 Multithreading Design*

Threads were used for the parallel design of the project. Threads are lightweight processes: they work on the same memory space and require less time to communicate (Silberschatz, Galvin, and Gagne, 2004). Since our model will run in a multi-core environment with constrained memory, threads are more suitable for parallel modeling.

Our project requires smaller tasks and an integrated environment, for which threads are the most efficient option.

As depicted in the parallel design in Figure 4-10, the *copymask* function initiates three threads of *separatemask*. Each thread executes the task of separating the contours of different thresholds: nucleus, cytoplasm and cell. Each *separatemask* function then will spawn a number of *handleobject* threads according to the number of contours in the input mask image. For n number of contours, *separatemask* will create n *handleobject* threads. Subsequently, the *handleobject* thread will create a TAS thread for each of the 3 channels (Channels 1, 2 and 3) simultaneously to compute the threshold adjacency statistics. By analyzing the design, it is obvious that the highest level of parallelism can be achieved at the last level: the TAS process. Therefore, the experimental results in this project focus mainly on TAS level parallelism.

#### ***4.3.3.3 Design of Shared Memory and Synchronization***

Communication between the threads was done by using shared memory. As a synchronization mechanism, *semaphore* was used to access the shared memory and communicate between the threads (Tanenbaum, 2007). The design uses a global variable ‘counter’ in the *separatemask* function, and increases the value as each contour mask is created. This counter value is also shared by the *handleobject* process to keep track of the contour it is operating on. This global variable, counter, executes as a critical section using *semaphore*.

The writing of an output file is also synchronized by the threads. Each thread writes the nine TAS values asynchronously. As one TAS thread finishes writing on the output file, it signals the next waiting thread to write on the output file. Furthermore, loading and releasing of images on threads is also done synchronously.

#### *4.3.3.4 Code Optimization*

The algorithm is also optimized to reduce the speed. The optimization occurs in the *handleobject* function, where a logical 'AND' operation between the mask image and the input image is executed. The operation iterates through the full 672x508 sized matrices in order to identify the targeted object. However, only a small subset of the matrix, the pixel values of the region of interest, is required for the computation. Therefore, to optimize this operation, the coordinates of the regions of interest are extracted, and the logical 'AND' operation is only applied to the region of interest of the image, thus reducing the iteration time.

### 4.3.4 Features of the Code

- The code is modularized: each function executes a specific task. As a result, it is easily maintainable and flexible for future modification.
- The model is threading-safe. Standard synchronization design procedures were followed to maintain deadlock proof.
- The software is documented and indented according to standard coding guidelines.

- The code uses memory efficiently and also allocates and releases memory properly to avoid memory leaks.

# 5 Result, Discussion and Conclusion

In this Chapter result will be presented to justify our findings with parallel scalability tests. Our software output will be validated against the Acapella output, and comparison of execution time against Acapella will be provided. Then a statistical representation of sample data and execution time will be given. A conclusion is drawn with suggestions for future work.

## 5.1 Performance Analysis of Parallelism

In order to evaluate the efficiency of parallelism, the code was tested in multi-core environment. The purpose of this evaluation was to measure the performance of the software with respect to parallelization. Accordingly, a scalability test was performed to evaluate the performance of the parallel system. Two common scalability tests were performed: strong scalability, to measure the performance of the software with a constant load as resources increase; and weak scalability, measuring the performance as load increases with the resources being constant (Kumar, Vipin, 1994).

The scalability test was initiated by using images with a single object as the input, then repeatedly processing the image using an increasing number of cores, from single to four cores. The image processing time for each case is subsequently recorded. The full test is then repeated by using images with an increasing number of objects, and processing each of the images with an increasing number of cores. The test was carried out by AMD CodeAnalyst performance analyzer running the programs on computer with an AMD Phenom Quad Core processor, 2.11 GHz, 4 GB memory, 64 bit Windows 7. Each individual image was run multiple times (four runs) and the computation times were then averaged.

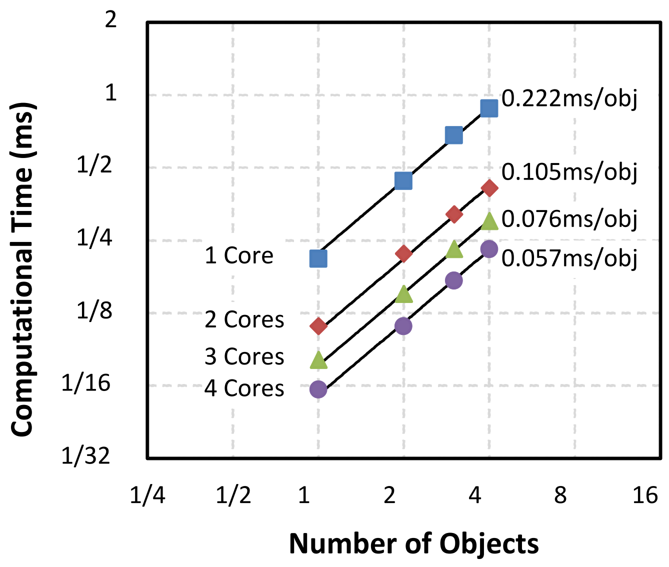


**Figure 5-1: Experimental results: Computational time measured against number of objects in the image using different numbers of cores.**

The graph in Figure 5-1 represents the computational time with increasing number of objects for four cores. The computational time is proportional to the number of objects. These findings also corroborate with Fahim et al (2011), who tested optimized GLCM code on multiple processors and compared to serial processor execution time. The graph also shows that the coefficient of proportionality, the slope of the curves, reduces from .222 ms/obj to .057 ms/obj from singe core to 4 cores. This indicates that the processing speed increases 4 folds from single core serial processing to multi-core parallel computation.

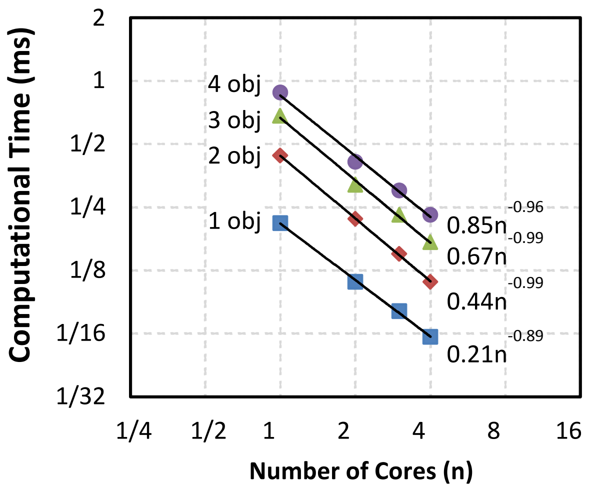


**Figure 5-2 Experimental results: Computational time measured against number of cores used using different numbers of objects in the image.**

Figure 5-2 represents a complementary comparison of computational time of four objects with respect to increasing number of cores. In this experiment, the cores remained constant and the load of the images was increased by adding objects to the image. The computational time of four objects is inversely proportional to increasing number of cores, which means the computational time reduces with an increasing number of cores for all three curves. The exponent in the power law trends of all three objects is very close to -1, indicating that the computational time is reciprocally related to the number of cores.

However, to analyze how well the program performs parallel compare to a single processor, a speedup test was performed on the datasets. Speedup is a well-known standard measurement metric for assessing the performance of parallelism. This measurement provides information of how ideal the parallel processing is by comparing the execution time of the program while using one core to multiple cores (Brown, 2000). The calculation is done by dividing the execution time on one processor core to *n* processor cores. A higher speedup value indicates more parallelism. As observed in the graph of Figure 5-4, the metric indicates significant improvement in the level of parallelism: four times more speedup than a single processor in all cases. Below is the calculation of speed up follows with the graph:

$$\text{Speedup} = \frac{\text{execution time of one processor}}{\text{execution time on n processors}}$$

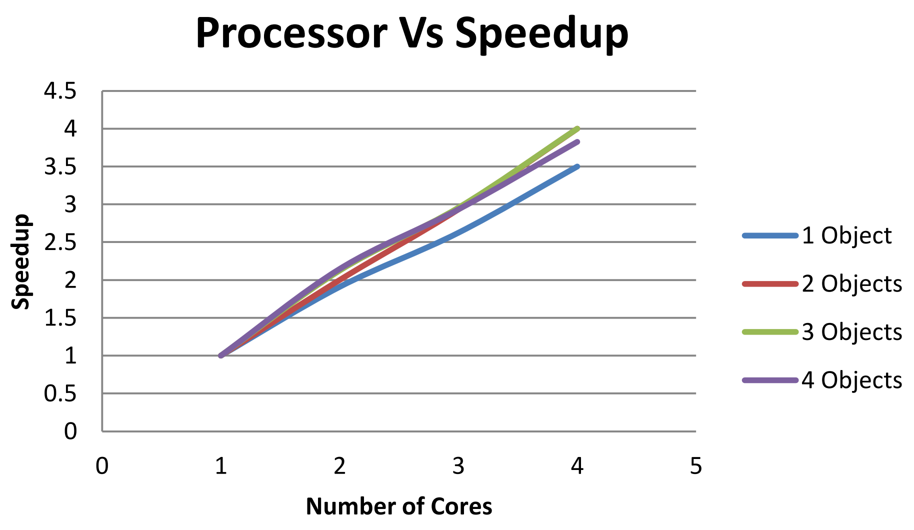


**Figure 5-3: Performance comparison: Speedup versus number of cores. Increase in speedup indicates increase in level of parallelism.**

It is also seen from the figure 5-4 that the speedup varies at 4 cores from 3.5 to 4 depending on the number of objects. Object 3 at core 4 has the most speed up value and provides the highest level of parallelism than other three objects. The speedup of the other three objects declines slightly from 3.5 to 4 with increasing cores. This variation on speedup values could be justified by three reasons. Firstly, a major portion of the software is I/O bound tasks, performing image load, read or writes; as a result, as the number of objects is increases more I/O operations were performed, slowing down the parallelization efficiency (Silberschatz et al., 2004). Secondly, not all the codes are parallelized, just thirty percent of the code, the remaining serial portion of the code decreases the performance of parallelization as the number of object increases. Finally, for every new object a thread is created. As object number increases the thread number increases accordingly. Increasing threads will take more time on synchronization and communication resulting on more processing time, therefore affecting the overall computational time (Silberschatz et al., 2004).

## 5.2 Validation of the Result

In order to validate the software's results, the output of the code was compared with Acapella's output in Table 5-1. Acapella has been used in many experiments in the McMaster Biophotonics Facility, so the results are authenticated and provide a credible dataset to validate our software. The output of Acapella and C code is provided below from a set of images. Although there is a negligible difference in the results in a few instances, this is due to the scripting language limitation of Acapella's API. Like any other scripting language, Acapella is limited to its supported specifications, such as routines, variables and objects. Consequently, due to Acapella being a less customizable script, complex algorithms are often very hard to implement, and accuracy can be compromised significantly. However, due to the flexibility of low-level languages, the final results of the C code are more precise and accurate,  even though both languages follow the same calculation of TAS by Hamilton (Hamilton et al., 2007a).

Result Comparison

| Channel | Object Number | Threshold | | TAS_01_20 | TAS_02_20 | TAS_03_20 | TAS_04_20 | TAS_05_20 | TAS_06_20 | TAS_07_20 | TAS_08_20 | TAS_09_20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | Cytoplasm 1 | Threshold 1 | Acapella | 0.0066462 | 0.0158487 | 0.047546 | 0.0864008 | 0.122699 | 0.163088 | 0.197853 | 0.175358 | 0.18456 |
| | | | C code | 0.0066462 | 0.0158487 | 0.047546 | 0.0864008 | 0.122699 | 0.163088 | 0.197853 | 0.175358 | 0.18456 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.003876 | 0.00599013 | 0.0158562 | 0.030303 | 0.0461593 | 0.0852713 | 0.0930233 | 0.143411 | 0.57611 |
| | | | C code | 0.003876 | 0.00599013 | 0.0158562 | 0.030303 | 0.0461593 | 0.0852713 | 0.0930233 | 0.143411 | 0.57611 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0038437 | 0.0102498 | 0.0307495 | 0.0621397 | 0.0916079 | 0.140935 | 0.124279 | 0.160794 | 0.3754 |
| | | | C code | 0.0038437 | 0.0102498 | 0.0307495 | 0.0621397 | 0.0916079 | 0.140935 | 0.124279 | 0.160794 | 0.3754 |
| 3 | Cytoplasm 1 | Threshold 1 | Acapella | 0.0002828 | 0.000848416 | 0.0042421 | 0.0084842 | 0.0483597 | 0.0927602 | 0.0854072 | 0.116799 | 0.642817 |
| | | | C code | 0.0002828 | 0.000848416 | 0.0042421 | 0.0084842 | 0.0483597 | 0.0927602 | 0.0854072 | 0.116799 | 0.642817 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0 | 0.000250564 | 0.0007517 | 0.0027562 | 0.0298171 | 0.0643949 | 0.0438487 | 0.0699073 | 0.788274 |
| | | | C code | 0 | 0.000250564 | 0.0007517 | 0.0027562 | 0.0298171 | 0.0643949 | 0.0438487 | 0.0699073 | 0.788274 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.00052 | 0.00520021 | 0.0171607 | 0.0364015 | 0.0785231 | 0.126885 | 0.117005 | 0.152886 | 0.465419 |
| | | | C code | 0.0010499 | 0.00419948 | 0.0183727 | 0.0356955 | 0.0839895 | 0.126509 | 0.120735 | 0.149606 | 0.459843 |
| 2 | Cytoplasm 2 | Threshold 1 | Acapella | 0.0020756 | 0.014944 | 0.0402657 | 0.0805313 | 0.15193 | 0.188045 | 0.173931 | 0.199668 | 0.148609 |
| | | | C code | 0.0020585 | 0.0148209 | 0.0395224 | 0.0794566 | 0.145327 | 0.186908 | 0.180733 | 0.198024 | 0.153149 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.0010419 | 0.00547017 | 0.0101589 | 0.0208388 | 0.0552227 | 0.092472 | 0.0989841 | 0.151602 | 0.564209 |
| | | | C code | 0.0010419 | 0.00547017 | 0.0101589 | 0.0208388 | 0.0552227 | 0.092472 | 0.0989841 | 0.151602 | 0.564209 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0038494 | 0.00769889 | 0.0162532 | 0.0491873 | 0.0919589 | 0.143713 | 0.130026 | 0.150984 | 0.40633 |
| | | | C code | 0.0034647 | 0.00606323 | 0.0181897 | 0.049372 | 0.0922477 | 0.145518 | 0.127761 | 0.151581 | 0.405803 |
| 3 | Cytoplasm 2 | Threshold 1 | Acapella | 0.0017534 | 0.00204559 | 0.0087668 | 0.028346 | 0.075979 | 0.125073 | 0.120982 | 0.158387 | 0.478667 |
| | | | C code | 0.0017534 | 0.00204559 | 0.0087668 | 0.028346 | 0.075979 | 0.125073 | 0.120982 | 0.158387 | 0.478667 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.0006509 | 0.000216967 | 0.0015188 | 0.0086787 | 0.0364504 | 0.0700803 | 0.0598828 | 0.0902582 | 0.732263 |
| | | | C code | 0.0006509 | 0.000216967 | 0.0015188 | 0.0086787 | 0.0364504 | 0.0700803 | 0.0598828 | 0.0902582 | 0.732263 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0024691 | 0.00329218 | 0.0131687 | 0.0304527 | 0.0987654 | 0.14856 | 0.112757 | 0.155144 | 0.435391 |
| | | | C code | 0.0024691 | 0.00329218 | 0.0131687 | 0.0304527 | 0.0987654 | 0.14856 | 0.112757 | 0.155144 | 0.435391 |
| 2 | Neuclus 1 | Threshold 1 | Acapella | 0.0024184 | 0.0157195 | 0.0411125 | 0.108827 | 0.159613 | 0.182588 | 0.185006 | 0.135429 | 0.169287 |
| | | | C code | 0.0059022 | 0.00927487 | 0.0451096 | 0.102445 | 0.150084 | 0.194351 | 0.190556 | 0.167791 | 0.134486 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.0008271 | 0.00413565 | 0.006617 | 0.0314309 | 0.063689 | 0.129032 | 0.110008 | 0.124897 | 0.529363 |
| | | | C code | 0.0034667 | 0.0032 | 0.0093333 | 0.0234667 | 0.0402667 | 0.0696 | 0.0885333 | 0.134667 | 0.627467 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0056657 | 0.00708215 | 0.0311615 | 0.0651558 | 0.117564 | 0.13881 | 0.124646 | 0.145892 | 0.364023 |
| | | | C code | 0.0043085 | 0.00689358 | 0.0159414 | 0.0366221 | 0.0779836 | 0.130116 | 0.125377 | 0.140026 | 0.462732 |
| 3 | Neuclus 1 | Threshold 1 | Acapella | 0.0007911 | 0.00237342 | 0.005538 | 0.0205696 | 0.0490506 | 0.121044 | 0.148734 | 0.174051 | 0.477848 |
| | | | C code | 0.0002139 | 0.000855432 | 0.0040633 | 0.011976 | 0.0457656 | 0.0904619 | 0.100941 | 0.134944 | 0.610778 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0 | 0.000662252 | 0.0013245 | 0.0046358 | 0.0324503 | 0.0827815 | 0.0781457 | 0.101325 | 0.698676 |
| | | | C code | 0.0001835 | 0.000183453 | 0.0016511 | 0.0066043 | 0.0229316 | 0.0432948 | 0.0398092 | 0.0647588 | 0.820583 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0014265 | 0.00713267 | 0.0242511 | 0.0527817 | 0.116976 | 0.17689 | 0.0998573 | 0.122682 | 0.398003 |
| | | | C code | 0.0024361 | 0.00527812 | 0.0133983 | 0.047503 | 0.091758 | 0.120585 | 0.112058 | 0.146569 | 0.460414 |
| 2 | Neuclus 2 | Threshold 1 | Acapella | 0.0147368 | 0.0273684 | 0.0378947 | 0.0673684 | 0.145263 | 0.128421 | 0.157895 | 0.168421 | 0.252632 |
| | | | C code | 0.0043415 | 0.010492 | 0.0416064 | 0.0846599 | 0.138205 | 0.171491 | 0.166787 | 0.187048 | 0.195369 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.0094086 | 0.016129 | 0.0241935 | 0.030914 | 0.077957 | 0.104839 | 0.0873656 | 0.133065 | 0.516129 |
| | | | C code | 0.0019694 | 0.00350109 | 0.0091904 | 0.0183807 | 0.045733 | 0.0741794 | 0.0787746 | 0.128446 | 0.639825 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0048426 | 0.00968523 | 0.0290557 | 0.0338983 | 0.123487 | 0.1477 | 0.101695 | 0.1477 | 0.401937 |
| | | | C code | 0.0041958 | 0.00804196 | 0.0143357 | 0.0391608 | 0.0832168 | 0.11993 | 0.112587 | 0.158042 | 0.46049 |
| 3 | Neuclus 3 | Threshold 1 | Acapella | 0 | 0 | 0.0030675 | 0.0322086 | 0.0705521 | 0.107362 | 0.0904908 | 0.136503 | 0.559816 |
| | | | C code | 0.0016047 | 0.00561647 | 0.0080235 | 0.0243381 | 0.0722118 | 0.131319 | 0.122225 | 0.145761 | 0.488901 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0 | 0 | 0.0021787 | 0.0174292 | 0.0511983 | 0.083878 | 0.0555556 | 0.0991285 | 0.690632 |
| | | | C code | 0.000747 | 0.00149393 | 0.0018674 | 0.0056022 | 0.0291317 | 0.0567694 | 0.0435107 | 0.0702148 | 0.790663 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0 | 0.00210084 | 0.0063025 | 0.0210084 | 0.102941 | 0.157563 | 0.102941 | 0.142857 | 0.464286 |
| | | | C code | 0.0012512 | 0.00406631 | 0.0050047 | 0.0184548 | 0.0791367 | 0.108539 | 0.0985299 | 0.156084 | 0.528933 |
| 3 | Neuclus 2 | Threshold 1 | Acapella | 0.0031898 | 0.0183413 | 0.0430622 | 0.0614035 | 0.0972887 | 0.133174 | 0.118022 | 0.185805 | 0.339713 |
| | | | C code | 0.0072243 | 0.0117871 | 0.0285171 | 0.0634981 | 0.102662 | 0.157034 | 0.175285 | 0.203422 | 0.25057 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0.001836 | 0.0134639 | 0.0312118 | 0.0367197 | 0.0771114 | 0.109547 | 0.0850673 | 0.127907 | 0.517136 |
| | | | C code | 0.0034474 | 0.00541738 | 0.0118197 | 0.0201921 | 0.0381679 | 0.0598375 | 0.0726422 | 0.0948042 | 0.693672 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0169051 | 0.0104031 | 0.023407 | 0.0507152 | 0.114434 | 0.135241 | 0.118336 | 0.149545 | 0.381014 |
| | | | C code | 0.0031323 | 0.00587314 | 0.0109632 | 0.032498 | 0.0712608 | 0.104933 | 0.110023 | 0.139389 | 0.521926 |
| 3 | Neuclus 2 | Threshold 1 | Acapella | 0 | 0.00171233 | 0.0011416 | 0.0085616 | 0.0348174 | 0.0456621 | 0.0525114 | 0.086758 | 0.768836 |
| | | | C code | 0.0009918 | 0.000743863 | 0.0039673 | 0.0143814 | 0.0513266 | 0.104885 | 0.0984379 | 0.12844 | 0.596826 |
| | | Threshold 2 | | | | | | | | | | |
| | | | Acapella | 0 | 0.0004914 | 0.0004914 | 0.0034398 | 0.029484 | 0.0481572 | 0.0378378 | 0.0687961 | 0.811302 |
| | | | C code | 0.0001988 | 0.000397693 | 0.0007954 | 0.0045735 | 0.0284351 | 0.0554782 | 0.0419567 | 0.0517001 | 0.816464 |
| | | Threshold 3 | | | | | | | | | | |
| | | | Acapella | 0.0040215 | 0.0147453 | 0.0227882 | 0.0254692 | 0.103217 | 0.130027 | 0.103217 | 0.119303 | 0.477212 |
| | | | C code | 0.0014075 | 0.00387051 | 0.0087966 | 0.0246305 | 0.0647431 | 0.104152 | 0.096411 | 0.110837 | 0.585151 |

**Table 5-1: Comparison of results of C code with Acapella.**

## 5.3 Execution Time Comparison

Two types of execution time were tested and are shown in Table 5-2. The program was first tested with converted TIFF images as input, not including the conversion time from flex to TIFF images (3rd column), and the second test was performed on FLEX images, including the conversion time from flex to TIFF (4th column). Figure 5-6 also provides the plot of Table 5-2.

| **Number of FLEX Images** | **Acapella** | ***C code (TIFF images as input)** | ****C code(FLEX images as input)** |
|---|---|---|---|
| 1 | 61 seconds | 7 seconds | 50 seconds |
| 6 | 350 seconds | 50 seconds | 245 seconds |
| 10 | 620 seconds | 105 seconds | 360 seconds |
| 20 | 1200seconds | 150 seconds | 780 seconds |
| 384 | 25800sec (7hour) | 550 seconds | 18010 seconds (5 hour) |

**Table 5-2: Execution time comparison of Acapella and C code.**
**only TIFF images as input, execution performed on converted images.*
*** conversion time from flex to TIFF + code execution on converted images.*

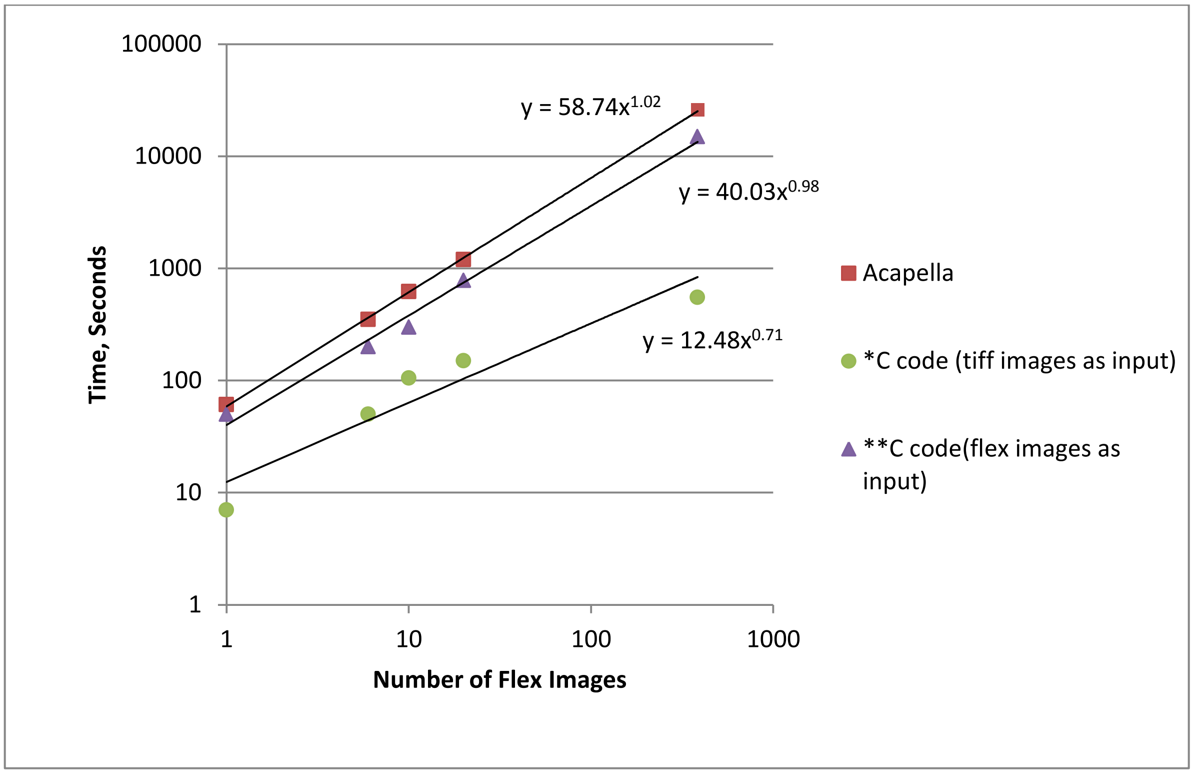


**Figure 5-4: The plot demonstrates the execution time between Acapella, C code with TIFF images and without TIFF images.**

As can be seen in the results above, the majority of the time of the C code is used on converting the FLEX images to TIFF format. This is because FLEX images are proprietary microscopy images and the only way to convert it to TIFF is to use the Bio-Format conversion tool package from the Laboratory for Optical and Computational Instrumentation (LOCI) (http://loci.wisc.edu/software/bio-formats) (Eliceiri et al., 2012). However, the Bio-Format conversion package from LOCI is not directly compliant with the C platform, as it is for MATLAB and other supported applications. The LOCI conversion tool, for C code, is a package of Java class libraries whose routines are used to convertFLEXimages and to extract their metadata. To make it compliant to C language, it uses the jar2lib program to generate a C++ proxy class for each equivalent Bio-Format Java class ("Bio-Formats | LOCI," n.d.). As a result, the conversion is very time consuming when using a language like C due to their indirect support through Java packages. On the other hand, conversion to TIFF images in Acapella is much faster and more efficient, since it uses their own proprietary conversion code, implemented directly to the flex images, and does not rely on third party Bio-Format tools. If conversion factor is not considered than the code operates much faster than the Acapella script, since the majority of the C code operation is spent converting the file with the Java tool. Consequently, the performance of the code can be appropriately evaluated without considering the conversion time. Regardless, in the future, Opera images will be in TIFF format instead of flex format.

## 5.4 Conclusion

The research focuses on modeling a parallel algorithm in C code that would extract textural features from HCS data. Most of the textural feature extraction algorithms used in HCS are computationally complex and intensive, requiring huge amount of time to process. Comparatively, Threshold Adjacency Statistics (TAS) by Hamilton et al. has proven to be more efficient in speed and accuracy than other algorithms. As well as, according to our review, this is the first work to date on parallelizing TAS and evaluating its performance. Scalability and speedup tests were performed on the model to evaluate the level of parallelism in the code. The running time and the quality of the code were also compared with the Acapella feature extraction software. Despite the limitation of the hardware (testing quad-core PC) where the performance evaluation test was performed, our model has provided better results than serial processing software.

## 5.5 Future Work

In this thesis we have used TAS features for parallelization, but other subcellular localization feature mentioned in section 2.3 needs to be explored on parallel modes in future. Furthermore, as seen from table 5.2 that the conversion from flex to tiff was very time consuming. This conversion could be performed in parallel to reduce the computational time. Another possibility could be operating on multiple images in parallel. Due to the limitation of the hardware, the program operates on one single input image at each run. If multiple images could be executed in parallel, the computational speed would immensely increase. But, this will require customized and expensive hardware.

Another directional approach could be applying TAS on other multicore hardware technologies, for example, FPGA and GPU. For further study, a comparative analysis of this parallel hardware would provide a better insight on their performances. In past FPGA and GPU technologies have been proven to provide better performance on texture and morphological features. In addition to the hardware, using advance optimization algorithm and data structure would also be an advantage to the processing speed. Advance data structure algorithm, such as link list, graph and hash table will also improve the execution time of the software. The developed software does not assign tasks to specific core. By using windows programming specific tasks could be assigned to specific core which will provide an equal distribution of load to all the cores. This will also optimize the overall use of cores.

Figure 4-10 in chapter four represents the feasible concurrent execution path of the software that could be executed in parallel. All the tasks between the parallel lines in the figure are independent, and could be executed in parallel. Due to time constraint, the thesis has only parallelized the code after segmentation. The optimal performance of the code could be further evaluated by parallelizing the segmentation tasks of the software. For further work, it is recommended to try different combination of concurrent execution path from figure 4-10, tasks that are between the parallel lines, to identify the optimal processing speed of the software.

# References

Bickle, M. (2010). The beautiful cell: high-content screening in drug discovery. *Analytical and bioanalytical chemistry*, *398*(1), 219–26. doi:10.1007/s00216-010-3788-3

Bio-Formats | LOCI.. Retrieved September 3, 2012, from http://loci.wisc.edu/software/bio-formats

Biostatus. (2012). Biostatus. Retrieved September 3, 2012 from http://www.biostatus.com/product/draq5/

Blaise Barney. (2012). Introduction to Parallel Computing. Retrieved June 20, 2012 from https://computing.llnl.gov/tutorials/parallel_comp/

Brown, R. G. (2000). Amdahl's Law & Parallel Speedup. Retrieved September 3, 2012, from http://www.phy.duke.edu/~rgb/brahma/Resources/als/als/node3.html

Canada, C. (2012). General cancer statistics at a glance. Retrieved September 3, 2012 from http://www.cancer.ca/

Canny, J. (1986). A computational Approach to edge detection. *IEEE Transactions on Computers*, *8*, 679–698.

Doyle, L. a, Ross, D. D., Sridhara, R., Fojo, a T., Kaufmann, S. H., Lee, E. J., & Schiffer, C. a. (1995). Expression of a 95 kDa membrane protein is associated with low daunorubicin accumulation in leukaemic blast cells. *British journal of cancer*, *71*(1), 52–8. Retrieved September 3, 2012 from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2033479&tool=pmcentrez&rendertype=abstract

Eliceiri, K. W., Berthold, M. R., Goldberg, I. G., Ibáñez, L., Manjunath, B. S., Martone, M. E., Murphy, R. F., et al. (2012). Biological imaging software tools. *Nature methods*, *9*(7), 697–710. doi:10.1038/nmeth.2084

Elmer, P. (2008). O PERA $^{TM}$ Software Manual. *Opera Software Manual*.

Fabbro, D., Ruetz, S., Buchdunger, E., Cowan-Jacob, S. W., Fendrich, G., Liebetanz, J., Mestan, J., et al. (2002). Protein kinases as targets for anticancer agents: from inhibitors to useful drugs. *Pharmacology & therapeutics*, *93*(2-3), 79–98. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/12191602

Faivre, S., Djelloul, S., & Raymond, E. (2006). New paradigms in anticancer therapy: targeting multiple signaling pathways with kinase inhibitors. *Seminars in oncology*, *33*(4), 407–20. doi:10.1053/j.seminoncol.2006.04.005

Fotiadis, D. (2002). *Scattering and Biomedical Engineering*. Greece: World Scientific Publishihng Co. Ltd.

Gao, Q.-B., Jin, Z.-C., Wu, C., Sun, Y.-L., He, J., & He, X. (2009). Feature Extraction Techniques for Protein Subcellular Localization Prediction. *Current Bioinformatics*, *4*(2), 120–128. doi:10.2174/157489309788184765

Gasparri, F., Sola, F., Bandiera, T., Moll, J., & Galvani, A. (2008). High-content analysis of kinase activity in cells. *Combinatorial chemistry & high throughput screening*, *11*(7), 523–36. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/18694389

Ghoreschi, K., Laurence, A., & O'Shea, J. J. (2009). Selectivity and therapeutic inhibition of kinases: to be or not to be? *Nature immunology*, *10*(4), 356–60. doi:10.1038/ni.1701

Gonzalez, R. C. (2008). *Digital Image Processing*, (third.). New Jersey: Prentice Hall.

Grama, Aananth, Gupta, Anshul, Karypis, G. (2003). *Introduction to Prallel Computing* (second.). ACM Press. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:[+Team+LiB+]+?#6

*Greco, J. (2005). Parallel image processing and computer vision architecture,* undergraduate thesis, University of Florida, Florida.

Hamilton, N. a, Pantelic, R. S., Hanson, K., & Teasdale, R. D. (2007a). Fast automated cell phenotype image classification. *BMC bioinformatics*, *8*, 110. doi:10.1186/1471-2105-8-110

Hamilton, N. a, Pantelic, R. S., Hanson, K., & Teasdale, R. D. (2007b). Fast automated cell phenotype image classification. *BMC bioinformatics*, *8*, 110. doi:10.1186/1471-2105-8-110

Hamilton, N. a, Wang, J. T. H., Kerr, M. C., & Teasdale, R. D. (2009). Statistical and visual differentiation of subcellular imaging. *BMC bioinformatics*, *10*, 94. doi:10.1186/1471-2105-10-94

Haney, S. A. (Ed.). (2008). *High Content Screening*. Hoboken, NJ, USA: John Wiley & Sons, Inc. doi:10.1002/9780470229866

Haralick, R.M.K. Shanmuga, I. D. (1973).Texture features for imageclassifiactaion. *.IEEE transaction system and management*, *SMC3*, 610–621.

Invitrogen. (2012). Invitrogen. Retrieved September 3, 2012 from http://products.invitrogen.com/ivgn/product/D1199

M. Tuceryan and A. K. Jain, ``Texture Analysis,'' *In The Handbook of Pattern Recognition and Computer Vision (2nd Edition*), by C. H. Chen, L. F. Pau, P. S. P. Wang (eds.), pp. 207-248, World Scientific Publishing Co., 1998. (Book Chapter) Kao, W. Y., Davis, C. E., Kim, Y. I., & Beach, J. M. (2001). Fluorescence emission spectral shift measurements of membrane potential in single cells. *Biophysical journal*, *81*(2), 1163–70. doi:10.1016/S0006-3495(01)75773-6

Karaman, M. W., Herrgard, S., Treiber, D. K., Gallant, P., Atteridge, C. E., Campbell, B. T., Chan, K. W., et al. (2008). A quantitative analysis of kinase inhibitor selectivity. *Nature biotechnology*, *26*(1), 127–32. doi:10.1038/nbt1358

Kheirkhah, F. M., & Haghipour, S. (2010). Classification of Subcellular Location Patterns in Fluorescence Microscope Images Based on Modified Threshold Adjacency Statistics. *Biomedical Engineering*, 1–7.

Kumar, Vipin, G. A. (1994). Analyzing scalibility of parallel algorithms.pdf. *Journal of Parallel And Distributed Computing*, *22*, 379–391.

Levsky, J. M., & Singer, R. H. (2003). Gene expression and the myth of the average cell. *Trends in cell biology*, *13*(1), 4–6. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/12480334

Liu, F. (2012). Contributions to Statistical Image Analysis for High Content Screening. PhD thesis, University of Michigan, Michigan

LOCI. (2012). BioFormat. Retrieved September 3, 2012 from http://loci.wisc.edu/software/bio-formats

Lu, Y., Lu, J., Liu, T., & Yang, J. (2010). Automated Cell Phase Classification for Zebrafish Fluorescence Microscope Images. *2010 20th International Conference on Pattern Recognition*, 2584–2587. doi:10.1109/ICPR.2010.633

Mcandrew, A. (2004). An Introduction to Digital Image Processing with Matlab Notes for SCM2511 Image Processing 1 Semester 1 , 2004. *Image Processing*.

Messerli, V. (1998). Tools for Parallel I / O and Compute Intensive Applications, *1915*.

Mooney, L. M., Al-Sakkaf, K. a, Brown, B. L., & Dobson, P. R. M. (2002). Apoptotic mechanisms in T47D and MCF-7 human breast cancer cells. *British journal of cancer*, *87*(8), 909–17. doi:10.1038/sj.bjc.6600541

Murphy, R. F., Velliste, M., & Porreca, G. (2003). Robust Numerical Features for Description and Classification of Subcellular Location Patterns in Fluorescence Microscope Images ∗. *Neural Networks*, 311–321.

N, O. (1979). Threshold Selection Method Form Gray-level Histogram". *IEEE Trans SMC*, 62–66.

Nanni, L., & Lumini, A. (2008). A reliable method for cell phenotype image classification. *Artificial intelligence in medicine*, *43*(2), 87–97. doi:10.1016/j.artmed.2008.03.005

Niederlein, A., Meyenhofer, F., White, D., & Bickle, M. (2009). Image analysis in high-content screening. *Combinatorial chemistry & high throughput screening*, *12*(9), 899–907. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/19531001

Oberholzer, M., Ostreicher, M., Christen, H., & Brühlmann, M. (1996). Methods in quantitative image analysis. *Histochemistry and cell biology*, *105*(5), 333–55. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/8773570

Peng, H. (2008). Bioimage informatics: a new area of engineering biology. *Bioinformatics (Oxford, England)*, *24*(17), 1827–36. doi:10.1093/bioinformatics/btn346

Pepperkok, R., & Ellenberg, J. (2006). Microscopy for systems biology. *Group*, *7*(September), 690–696.

Rausch, O. (2006). High content cellular screening. *Current opinion in chemical biology*, *10*(4), 316–20. doi:10.1016/j.cbpa.2006.06.004

Ronneberger, O., Baddeley, D., Scheipl, F., Verveer, P. J., Burkhardt, H., Cremer, C., Fahrmeir, L., et al. (2008). *Spatial quantitative analysis of fluorescently labeled nuclear structures: problems, methods, pitfalls. Chromosome research : an international journal on the molecular, supramolecular and evolutionary aspects of chromosome biology* (Vol. 16, pp. 523–62). doi:10.1007/s10577-008-1236-4

Roumi, M. (2009). *MSc THESIS Implementing Texture Feature Extraction Algorithms on FPGA. Electrical Engineering*.

Secko, D. (2011). Protein phosphorylation: a global regulator of cellular activity. *The Sceince Creative Quarterly*, *6*.

Shahbahrami, A., Pham, T. A., & Bertels, K. (2011). Parallel implementation of Gray Level Co-occurrence Matrices and Haralick texture features on cell architecture. *The Journal of Supercomputing*, *59*(3), 1455–1477. doi:10.1007/s11227-011-0556-x

Silberschatz, A., Galvin, P. B., & Gagne, G. (2004). *Operating System Concepts, Seventh Edition* (p. 921). John Wiley & Sons. Retrieved from http://www.amazon.com/Operating-System-Concepts-Seventh-Edition/dp/0471694665

Starkuviene, V., & Pepperkok, R. (2007). The potential of high-content high-throughput microscopy in drug discovery. *British journal of pharmacology*, *152*(1), 62–71. doi:10.1038/sj.bjp.0707346

Tanenbaum, A. (2007). *Modern Operating System*. New jersey: Prentice Hall.

Wollman, R., & Stuurman, N. (2007). High throughput microscopy: from raw images to discoveries. *Journal of cell science*, *120*(Pt 21), 3715–22. doi:10.1242/jcs.013623

Wong, S. T. C. (2006). Informatics challenges of high-throughput microscopy. *IEEE Signal Processing Magazine*, *23*(3), 63–72. doi:10.1109/MSP.2006.1628879

Yang, X., Beyenal, H., Harkin, G., & Lewandowski, Z. (2000). Quantifying biofilm structure using image analysis. *Journal of microbiological methods*, *39*(2), 109–19. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/10576700

Zanella, F., Lorens, J. B., & Link, W. (2010). High content screening: seeing is believing. *Trends in biotechnology*, *28*(5), 237–45. doi:10.1016/j.tibtech.2010.02.005

# Appendix A

## Statistical Analysis

A data set is being collected to understand the statistical properties of the whole population. A subset size is N=2,304 trials (N) of data and it was taken from Biophotonics Lab. The minimum, maximum and average numbers of cells found were 0, 180 and 53.47, respectively. The standard deviation of the data was σ=32.43. The histograms in Figure A-1 shows the frequency of occurrence (n) and normalized occurrence (n/N) versus the number of cells. The bin width of σ≈8. The probability distribution function, PDF=(n/N)/Δx, is also depicted in Figure 5-7. One observes in the figure that the distribution is skewed.

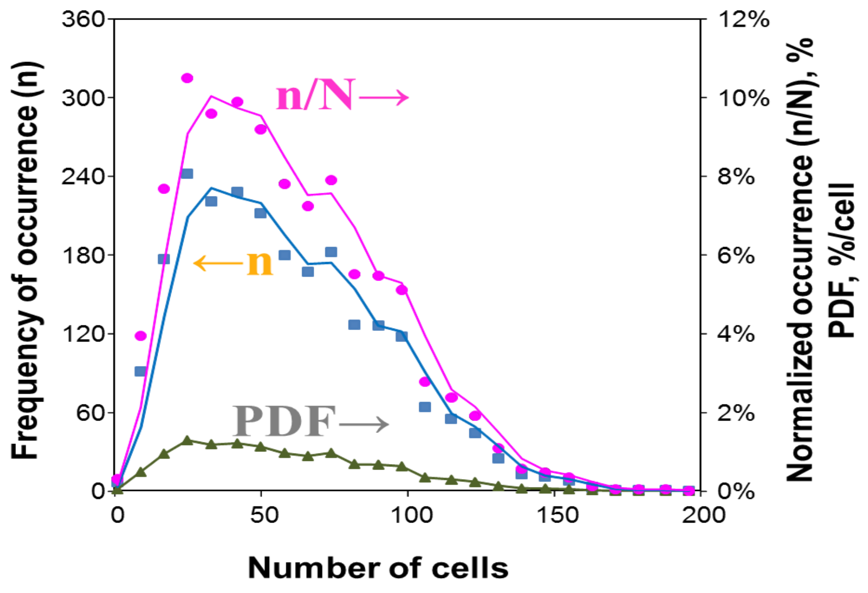


**Figure A-1: Histograms of Probability density function plot, shown alongside the frequency of occurrence of various cell counts.**

Figure A-2 compares the data distribution to the standard normal distribution. The blue curve corresponds to the normalized probability density function of the distribution deviated from the mean with 1 standard deviation and 0.25 bin width. The pink line represents the normal distribution.

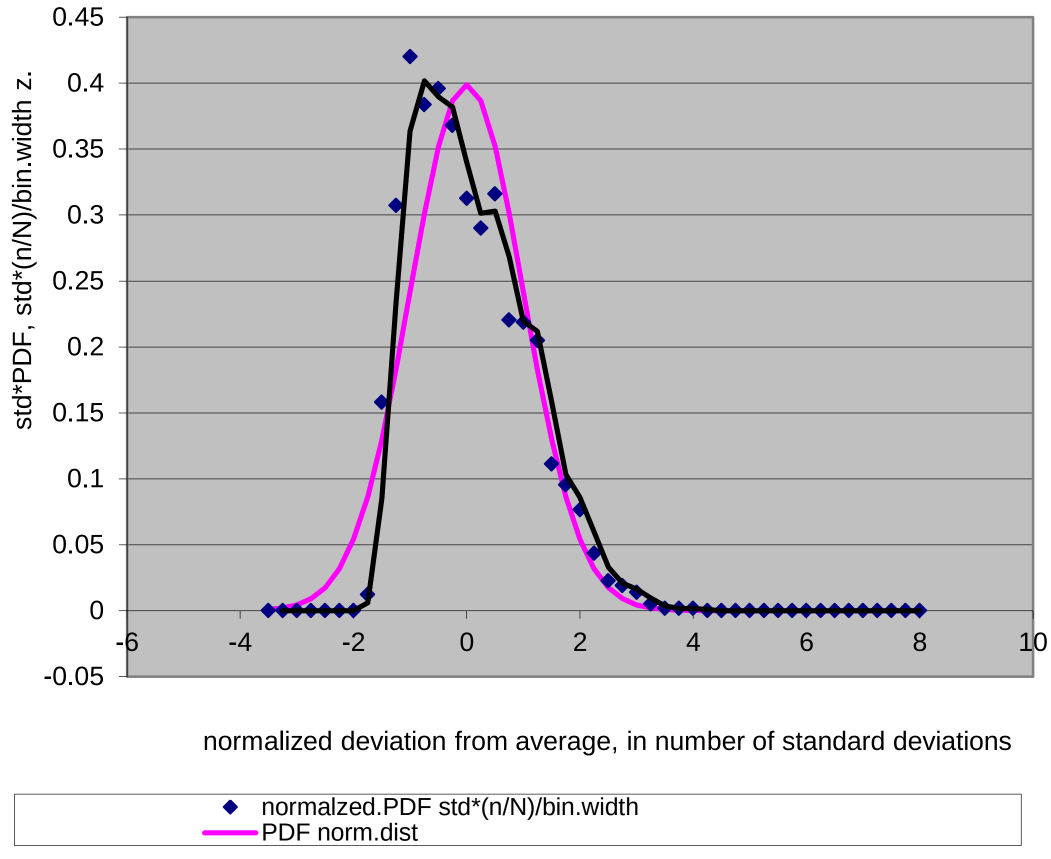


**Figure A-2: Probability density function of the experimental data plotted with closest matching to normal distribution**