Computer Aided Analysis of IHC and H&E Stained Histopathological Images in Lymphoma and Lupus

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Siddharth Samsi, B.E., M.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2012

Dissertation Committee:

Ashok K. Krishnamurthy, Advisor
Bradley Clymer
Kimerly Powell
Abstract

The use of computers in medical image analysis has seen tremendous growth following the development of imaging technologies that can capture image data in-vivo as well as ex-vivo. While the field of radiology has adopted computer aided image analysis in research as well as clinical settings, the use of similar techniques in histopathology is still in a nascent stage. The current gold standard in diagnosis involves labor-intensive tasks such as cell counting and quantification for disease diagnosis and characterization. This process can be subjective and affected by human factors such as reader bias and fatigue. Computer based tools such as digital image analysis have the potential to help alleviate some of these problems while also offering insights that may not be readily apparent when viewing glass slides under an optical microscope. Commercially available high-resolution slide scanners now make it possible to obtain images of whole slides scanned at 40x microscope resolution. Additionally, advanced tools for scanning tissue images at 100x resolution are also available. Such scanning tools have led to a large amount of research focused on the development of image analysis techniques for histopathological images. While the availability of high-resolution image data presents innumerable research opportunities, it also leads to several challenges that must be addressed.
This dissertation explores some of the challenges associated with computer-aided analysis of histopathological images. Specifically, we develop a number of tools for Follicular Lymphoma and Lupus. We aim to develop algorithms for detection of salient features in tissue biopsies of follicular lymphoma tumors. We analyze the algorithms from a computational point of view and develop techniques for processing whole slide images efficiently using high performance computing resources. In the application of image analysis for Lupus, we analyze mouse renal biopsies for characterizing the distribution of infiltrates in tissue as well as develop algorithms for identification of tissue components such as the glomeruli, which play a significant role in the diagnosis of the disease. Finally, we explore the development of a web-based system for dissemination of high-resolution images of tissues with the goal of advancing collaboration, research and teaching. Through the use of web technologies and developments in the field of geospatial imaging, we demonstrate the efficacy of an online tissue repository that can enable pathologists, medical students and all researchers to explore these images as well as use high performance computing systems to leverage computer-aided diagnosis tools in their field.
Dedicated to my family
Acknowledgments

This dissertation would not have been possible without the encouragement and support of many people. First and foremost, I would like to thank my wife, Dot for her unending patience and support through this process. Without her encouragement, it would have been significantly harder to complete my research. At the same time, I would like to thank our kids Uma and Ari for being the stress relievers during those times that I needed it the most. Hanging out with them gave me the much needed break and laughter that made the whole journey worth it. I would also like to thank my parents, without whose sacrifices I would not have been able to pursue my education at the Ohio State University. They have always been encouraging and supportive of all my academic pursuits and taught me the value of hard work and perseverance.

I would also like to thank my advisor, Dr. Krishnamurthy for his support throughout these many years that I have been at OSU. He always gave me opportunities to pursue very interesting areas of research that have helped me grow in my academic as well as professional life. I would like to thank Prof. Bradley Clymer not just for being on my Ph.D. committee, but also for making me think along very unique directions. His questions about my research have always led me to new and interesting insights. I would also like to express my deep gratitude to Dr. Kimerley Powell for being on
my committee. Working with her has given me a different perspective on the real-world applications of medical imaging research. I would like to express my gratitude to Dr. Metin Gurcan, who introduced me to histopathological image analysis and gave me my first research project in this area. I would not have been involved in this challenging and exciting field without his help.

I would be remiss if I did not mention Dr. John Nehrbass, who I consider to be a good friend and a mentor. Working with him over the years has taught me many valuable lessons that I hope I can pass on to others. He has always been a voice of support, encouragement and the occasional discipline when I needed it most.

Finally, I would like to thank my colleagues at the Ohio Supercomputer Center who have helped me in various ways. I would like to especially mention Don Stredney, who has given me wonderful insights into many aspects of medical imaging and human psychology and has constantly championed for me to focus on my research interests. I would like to thank Vijay Gadepally, Ben Smith, Brad Hittle, Dr. Thomas Kerwin and Dr. Judy Gardiner for putting up with my constant brainstorming sessions involving my research and for creating an environment that was conducive to research and exchange of ideas.
Vita

February, 1977 ......................... Born in Sangli, India

July, 1999 ............................... B.E. Electronics & Telecommunications,
                                             Pune University, Pune, India

March 2003 ............................ M.S. E.E.,
                                             Department of Electrical and Computer Engineering,
                                             The Ohio State University,
                                             Columbus, OH

March 2003 - February 2006 .......... Graduate Research Associate,
                                             Department of Electrical Engineering,
                                             The Ohio State University,
                                             Columbus, OH

March 2006 - present ................ Computational Science Researcher,
                                             Ohio Supercomputer Center,
                                             Columbus, OH

Publications

Research Publications


**Instructional Publications**


**Fields of Study**

Major Field: Electrical and Computer Engineering

Studies in:
- Signal Processing, Digital Speech Processing
- Signal Detection and Estimation Theory
- Communications Theory

Prof. Ashok K. Krishnamurthy
Prof. Philip Schniter
Prof. Oscar Y. Takeshita
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xv</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Contributions of this dissertation</td>
<td>4</td>
</tr>
<tr>
<td>2. Computer Aided Analysis of Tissue Biopsies of Follicular Lymphoma</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 Diagnosis and classification of follicular lymphoma</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2 Methodology</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Follicle detection in IHC images</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Rough segmentation of follicles using clustering</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Color feature</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2 Smoothing</td>
<td>13</td>
</tr>
<tr>
<td>2.3.3 Texture energy using co-occurrence matrix</td>
<td>15</td>
</tr>
<tr>
<td>2.3.4 Clustering</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Overlapping follicle separation</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Follicle boundary smoothing</td>
<td>19</td>
</tr>
<tr>
<td>2.6 Results</td>
<td>20</td>
</tr>
</tbody>
</table>
2.6.1 Measuring segmentation accuracy ........................................ 21
2.6.2 Sensitivity to processing parameters ................................. 24
2.7 Follicle Identification Using MALDI-MS ................................. 24
2.8 MALDI-MS data collection .................................................. 27
2.9 Image analysis .............................................................. 29
2.10 MALDI-MS data analysis ................................................... 31
   2.10.1 Methodology ......................................................... 33
2.11 Discussion .................................................................. 34

3. Semi-automated Quantification of IHC and H&E Stained Renal Biopsies of Lupus ......................................................... 37
   3.1 Quantification of IHC Staining in Images of Renal Biopsies ........ 38
   3.2 Proposed Approach ......................................................... 40
      3.2.1 Color Segmentation .................................................. 41
      3.2.2 Post Processing Segmentation Results ............................ 43
      3.2.3 Quantification of CD3 Staining ..................................... 45
   3.3 Results .................................................................. 48
   3.4 Glomeruli Detection in H&E Stained Mouse Renal Biopsies ....... 53
      3.4.1 Proposed Algorithm for Glomeruli Segmentation ............... 55
      3.4.2 Color Based Segmentation .......................................... 57
      3.4.3 Curve extraction ....................................................... 60
      3.4.4 Non-Convex Hull for Object Representation .................... 65
      3.4.5 Perceptual Organization ............................................. 70
   3.5 Results .................................................................. 77

4. Multiprocessor and GPGPU Computations for Efficient Analysis of Whole-Slide Images .................................................... 84
   4.1 Introduction ................................................................. 84
   4.2 Parallel Computing using MATLAB ..................................... 86
      4.2.1 Multi-threading in MATLAB ....................................... 86
      4.2.2 Multi-Processor MATLAB .......................................... 86
      4.2.3 Parallel computing tools for MATLAB ............................ 88
   4.3 Multi-processor approaches ............................................... 90
      4.3.1 Embarrassingly parallel approach ................................. 91
      4.3.2 Using distributed arrays ............................................. 93
      4.3.3 Fine grained parallelism ............................................. 96
      4.3.4 MATLAB on General Purpose Graphics Processing Unit .... 98
      4.3.5 Graphics Processing Units ......................................... 98
      4.3.6 Interfacing MATLAB with GPGPUs ............................... 99
      4.3.7 MATLAB plug-in for CUDA ....................................... 100
4.3.8 MATLAB Toolboxes for GPGPU computing .......................... 101
4.3.9 Considerations for GPGPU computing ............................. 101
4.3.10 The SSCA#3 application ............................................ 102
4.3.11 The SQIF Application ................................................ 106
4.3.12 Discussion .............................................................. 110

4.4 Analysis of whole slide tissue images ................................. 111
4.4.1 Challenges ............................................................... 112
4.4.2 Analysis of memory usage ............................................ 114
4.4.3 Processing whole slide image ........................................ 115
4.4.4 Overcoming memory limits .......................................... 116
4.4.5 Identifying and removing block artifact ........................... 117
4.4.6 Overcoming time limitations ........................................ 119

4.5 Parallel implementation of algorithm ................................. 120
4.5.1 Parallel 2D Filtering .................................................. 122
4.5.2 K-means On Distributed Matrices ................................. 124
4.5.3 Results ................................................................. 125
4.5.4 Parallel algorithm performance .................................... 128

4.6 Conclusion ................................................................. 132

5. Web Based Tools for Viewing and Annotation of Whole-Slide Images .... 134

5.0.1 Serving High Resolution Images over the Web .................... 135
5.0.2 Architecture ............................................................ 137
5.0.3 Implementation Using Google Maps ................................ 141
5.0.4 Annotating Histopathological Images ............................. 144

5.1 Results and Future Work ................................................. 148

6. Conclusion ................................................................. 152

Bibliography ................................................................. 155
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Recursive watershed algorithm for splitting merged regions</td>
</tr>
<tr>
<td>2.2</td>
<td>Similarity scores for 8 test images at 4x microscope resolution: Overall average score 87.11%</td>
</tr>
<tr>
<td>2.3</td>
<td>Confusion matrix as a percentage for leave one out validation using SVM classifier</td>
</tr>
<tr>
<td>2.4</td>
<td>Confusion matrix as a percentage for leave one out validation using KNN classifier</td>
</tr>
<tr>
<td>2.5</td>
<td>Confusion matrix as a percentage for leave one out validation using SVM classifier</td>
</tr>
<tr>
<td>2.6</td>
<td>Confusion matrix as a percentage for leave one out validation using KNN classifier</td>
</tr>
<tr>
<td>3.1</td>
<td>Calculation of co-circularity metric: Results</td>
</tr>
<tr>
<td>4.1</td>
<td>Calculating $\pi$: Sequential and parallel implementation using <code>parfor()</code></td>
</tr>
<tr>
<td>4.2</td>
<td>Parallelism using Distributed Arrays: Parallel 2D Convolution using pMATLAB and bcMPI</td>
</tr>
<tr>
<td>4.3</td>
<td>Parallelism using distributed arrays: Parallel 2D FFT using Star-P</td>
</tr>
<tr>
<td>4.4</td>
<td>An example of fine grained parallelism using bcMPI</td>
</tr>
<tr>
<td>4.5</td>
<td>Code Snippets for <code>genSARimage()</code>: Serial Version</td>
</tr>
</tbody>
</table>
4.6 Code Snippets for genSAImage(): Parallel Version using pMATLAB + bcMPI additions ........................................... 104
4.7 Algorithm run time before and after GPGPU porting ............... 105
4.8 GPGPU Data Transfer Overhead ................................... 106
4.9 Memory required for analysis of 4x resolution CD10 image .......... 113
4.10 Average computation time for various operations in the serial implementation of proposed algorithm: Timings are averaged over 5 runs for all images listed in Table 4.12 ........................................ 114
4.11 Memory usage for serial implementation when analyzing whole slide image of size 9,690 x 17,460 pixels ........................................... 115
4.12 Image sizes for data used in this study ................................ 125
4.13 Comparison of total algorithm time in seconds for serial algorithm with and without block processing and parallel algorithm running on 12 processors and using block processing ............................ 127
4.14 Data distribution across ten processors ................................ 128
5.1 Comparison of methods used to serve high resolution images over the internet using a single multi-resolution file .................. 142
5.2 Number of 256x256 pixel images created at each resolution for one of the images used in this research: A total of 73,681 images are generated for all resolutions combined ........................................ 145
5.3 Pseudo code for converting point in a Google Maps image tile to latitude and longitude values based on the Mercator projection ........ 149
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Follicles in CD10 and CD20 stained tissues: Images scaled down to 2x microscope resolution</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Adjacent tissue slices stained using H&amp;E and IHC stains at 2x microscope resolution</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Follicle regions in IHC stained tissue</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Outline of proposed algorithm for identifying follicles in CD20 stained tissue images</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Sample histogram of Hue channel from CD10 stained image</td>
<td>14</td>
</tr>
<tr>
<td>2.6 Example scatter plot and labeled image for test image after k-means clustering</td>
<td>16</td>
</tr>
<tr>
<td>2.7 Application of recursive watershed transform to split merged objects</td>
<td>18</td>
</tr>
<tr>
<td>2.8 Boundary smoothing using Fourier descriptors</td>
<td>20</td>
</tr>
<tr>
<td>2.9 Result of proposed algorithm for images using CD10 immunostaining</td>
<td>21</td>
</tr>
<tr>
<td>2.10 Hue channel histogram for different image content</td>
<td>23</td>
</tr>
<tr>
<td>2.11 MALDI-MS data acquisition procedure</td>
<td>26</td>
</tr>
<tr>
<td>2.12 H&amp;E stained tonsil tissue and corresponding MALDI-MS laser grid image</td>
<td>28</td>
</tr>
<tr>
<td>2.13 Identifying laser spot location on MALDI-MS grid: Image segmentation using Otsu [1] thresholding, edge detection and image morphology</td>
<td>30</td>
</tr>
</tbody>
</table>
2.14 Sample location of regions of interest in H&E stained tissue - Mantle (Red), Follicle (Green) and Intra-Follicle (Blue) regions.

2.15 Sample mass spectrum plots from representative mantle zone and follicle regions.

2.16 Illustration of challenges to MALDI-MS analysis: Laser spots from MALDI-MS acquisition may include spectra from multiple different regions.

3.1 Two different tissue samples stained with CD3 antibody stain: The protocol used can significantly affect the appearance of the tissue image.

3.2 Proposed algorithm for quantification of IHC stained tissue images of Lupus.

3.3 CMYK color conversion: Image from Fig. 3.1(a) converted to the CMYK colorspace.

3.4 CMYK color conversion: Image from Fig. 3.1(b) converted to the CMYK colorspace.

3.5 Yellow channel in CMYK: Tissues shown in Fig. 3.1 were converted to the CMYK colorspace. The yellow channel for both images and the corresponding histograms are shown in this figure.

3.6 Segmentation results: Otsu segmentation can be used to identify CD3-positive cells in the tissue.

3.7 Quantification of CD3 positive cell distribution: Concentric circles of increasing radius around selected locations in the tissue are used for quantifying the distribution of CD3 positive cells across the tissue.

3.8 Example of positive stained cell detection around selected glomeruli in two different tissues: Cells in region of interest are shown in green.

3.9 Example of positive stained cell detection around selected tubules in two different tissues: Cells in region of interest are shown in green.
3.10 Quantification of infiltrate around glomeruli: Number of positive cells found in concentric rings of 100µm around glomeruli in each tissue. Unique colors in each plot signify a different glomerulus in the tissue.

3.11 Quantification of infiltrate around tubules: Number of positive cells found in concentric rings of 100µm around selected tubules in each tissue. Unique colors in each plot signify a different tubule.

3.12 Alternative method for quantification of IHC-stained tissue images: Red regions depict glomeruli locations, blue indicates selected tubule locations and green lines indicate regions around a glomeruli or tubule to be associated with the specific object.

3.13 Mouse Glomerulus: Schematic diagram and a sample glomerulus section at 40x resolution. Schematic from [2]

3.14 Mouse glomeruli in H&E stained tissue: Regions of interest from tissue biopsies of two different mice are shown here. Glomeruli are highlighted in green.

3.15 Proposed algorithm for glomeruli detection in H&E stained mouse renal biopsies

3.16 Color-based segmentation of glomeruli in H&E stained tissue

3.17 Morphological cleanup: Removing small objects and non-curved objects in the thresholded image

3.18 Results of color based segmentation for region of interest extracted from four different tissue biopsies: Tissue 1 and 2

3.19 Results of color based segmentation for region of interest extracted from four different tissue biopsies: Tissue 3 and 4

3.20 Segmented objects and the desired curve representation of the objects shown in red

3.21 Curve extraction from objects: Segmented objects can be skeletonized to obtain representation in the form of 1-pixel thick curves.
3.22 Effect of non-smooth boundaries on the skeletonization of objects: Noise in object boundaries can lead to multiple branches in the object skeleton as illustrated here

3.23 Effects of boundary smoothing using median filter - Original object shown in red and smoothed version in blue

3.24 Illustration of different characteristic shapes generated from a set of points $P$

3.25 Calculation of non-convex hull for $\lambda_P = 0.5$: Red indicates the $\chi$ shape at each step and the dashed lines represent the removed edges of the triangulation

3.26 Characteristic shape for segmented object for varying $\lambda$

3.27 Optimal convex hull and the skeletons obtained from the resulting $\chi$ shape

3.28 Co-circularity derived from segments in an image

3.29 This figure illustrates one of the difficulties with using tangents to the end points of the curve

3.30 Forming tangents to curves at the given endpoints by circle fitting

3.31 Calculation of co-circularity metric: Tangents are constructed at the endpoints of the segments being paired

3.32 Directionality of a curve

3.33 Using directionality of a curve to find all objects that can potentially be grouped together

3.34 Example result: Objects in green and red were grouped by the perceptual grouping technique described here

3.35 Example result: Objects in red and green were grouped by the procedure described in Section 3.4.5
3.36 Example result: Objects in green, blue and yellow were grouped by the perceptual grouping technique described here ........................................... 79

3.37 Proposed approach fails to connect the disjoint objects shown here: Different colors represent objects that were not grouped together . . . 81

3.38 Thickening of objects at endpoints can cause errors in the grouping procedure ................................................................. 81

3.39 Co-circularity metric calculation and it’s failure to predict co-circularity .................................................. 82

3.40 Using convex hull to determine orientation and directionality of object .................................................. 82

4.1 Relative speedup using 16 threads on a 16 core, 4 socket system with the built-in multi-threading in MATLAB .............................................. 87

4.2 Architecture of pMATLAB/bcMPI from MIT Lincoln Laboratories and Ohio Supercomputer Center ................................................. 89

4.3 Architecture of Parallel Computing Toolbox™ and MATLAB® Distributed Computing Server from The MathWorks™ ....................... 90

4.4 Architecture of the Star-P system from Interactive Supercomputing (acquired by Microsoft Inc., in 2010) ................................................. 91

4.5 Multi-Processor vs. Sequential run time for 2D convolution: Performance of parallel 2D convolution on multi-processor system using pMATLAB and bcMPI ........................................... 95

4.6 Multi-Processor vs. Sequential run time for 2D FFT: Runtime curves obtained by varying problem size and parallelized using Star-P . . . 95

4.7 GPU architecture ................................................. 100

4.8 Multi-Processor Results for SSCA#3 application: Speedup of SSCA#3 for a fixed scale (6), and parallelized using bcMPI and pMATLAB . . 103

4.9 Parallelism Strategy for SQIF Application ................................................. 108

4.10 Multi-Processor Runtime for SQIF Application: For varying NSquids and using bcMPI + pMATLAB ................................................. 109

xix
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.11</td>
<td>Multi-Core run-time for SQIF Application: Runtime and speedup curves for varying NSquids and parallelized using the CUDA Mex interface compared with CPU based parallelization results</td>
</tr>
<tr>
<td>4.12</td>
<td>Detection of grid artifact in whole slide image: Artificially straight object borders indicate the presence of grid artifacts produced as a result of block processing of large image</td>
</tr>
<tr>
<td>4.13</td>
<td>Parallel implementation: Image is distributed across ( N_p ) processors and processed in ( B_w \times B_h ) blocks on each processor</td>
</tr>
<tr>
<td>4.14</td>
<td>Inter-processor communication: Data exchange between processors is indicated by the blue lines. Processors need to exchange borders columns/rows of data</td>
</tr>
<tr>
<td>4.15</td>
<td>Whole slide image analysis: Two tissues used in this study are shown here. Green outlines indicate follicle boundaries obtained from the proposed algorithm</td>
</tr>
<tr>
<td>4.16</td>
<td>Run-time for parallel median filter and texture energy calculation for two images used in the study</td>
</tr>
<tr>
<td>4.17</td>
<td>Run-time for parallel k-means calculations for two images used in the study</td>
</tr>
<tr>
<td>4.18</td>
<td>Average timing for parallel median filtering, texture energy calculation and k-means clustering for four CD10 stained tissue images used in the study</td>
</tr>
<tr>
<td>4.19</td>
<td>Computation time for parallel algorithm: Average time for entire algorithm calculated over 10 runs of the algorithm across all images used in the study</td>
</tr>
<tr>
<td>5.1</td>
<td>Whole slide image of size 103,767 ( \times ) 84,337 pixels at 40x microscope resolution: Green rectangle highlights the region of interest being displayed in a browser at higher resolution. Image courtesy of Dr. Christopher Pierson, OSU College of Medicine</td>
</tr>
<tr>
<td>5.2</td>
<td>Options for serving up histopathological images using a single image or multiple tiles</td>
</tr>
</tbody>
</table>
5.3 Example image: At 40x resolution this image is 58,224 x 61,879 pixels in size ................................................................. 142

5.4 Zoom level 1: Four sub-images of size 256x256 pixels. Actual tile size can be smaller depending on the size of the underlying image at each resolution ......................................................... 143

5.5 Zoom level 2: Sixteen sub-images of size 256x256 pixels .............. 144

5.6 Conversion of pixel co-ordinates to latitude and longitude values: Values are computed based on a 256x256 pixel image tile ...................... 145

5.7 Example of annotating histopathological images ....................... 147

5.8 Outlining results of image segmentation algorithm using Google Maps API: Paths around the glomeruli are shown in green, yellow and red . 150
Chapter 1: INTRODUCTION

1.1 Background

Histopathology is the examination of tissue under a microscope for disease characterization. In clinical practice, tissue biopsies from patients are processed and stained using a variety of stains in order to highlight various structures in the tissue. Commonly used stains include the Hematoxilin & Eosin (H&E) which stains the tissue in hues of blue and pink and immunohistochemical (IHC) stains such as CD3, CD10, CD20, CD21, CD68 which stain different parts of the tissue in brown and blue hues. By using different stains on tissue sections, pathologists can perform various diagnostic tasks such as counting the number of malignant cells, quantifying percentages of various cell types, the amount of cell activity, identifying specific structures (such as follicles) in the tissue, etc.

The current gold standard for the diagnosis of various types of cancer involves the manual examination of tissues under a microscope. This process can be tedious and error-prone and is also affected by reader bias. For example, the diagnosis of Hodgkin Lymphoma relies on the identification of double nucleated cells in a tissue sample containing hundreds of thousands of cells. The diagnosis and treatment of
Follicular Lymphoma relies on the number of centroblasts in ten representative follicles in a tissue biopsy. Studies have found high inter and intra-reader variability in the diagnosis of the same tissue sample for lymphomas [3–5]. Because of the central role of such diagnosis in the treatment of the disease, there is a need for additional, objective tools that the experts can use to alleviate the manual process.

Recent developments in imaging technology [6–8], have led to the availability of high resolution slide scanners that can scan and digitize histopathology slides at magnifications of 40x microscope resolution using dry slides and up-to 80x using oil immersions [6]. The availability of such technology has led to a significant amount of research into the analysis of these images in order to provide a quantitative assessment of the disease [9–11]. Image analysis tools have been developed for the automated grading of neuroblastoma [12, 13], follicular lymphoma [14, 15], breast cancer [16, 17] and prostate cancer [18, 19], along with other types of cancer. While most of the current research has focused on cancer, other diseases such as Lupus, Sjögren’s disease, etc. can also benefit from these developments since the same principles of histopathology apply in the diagnosis and treatment process.

The approaches to computer aided diagnosis of histopathology images involve segmentation of the image into relevant structures, feature extraction, classification and quantification. A wide range of approaches have been proposed for the segmentation of medical images [9, 20, 21]. One of the most common approach is the use of textures for image segmentation. Implementations of texture based segmentation include the use of co-occurrence matrix based features proposed by Haralick [22], local binary patterns [23,24], fractals [25] and Bayesian methods for texture segmentation.
Co-occurrence based methods were used by [15] and [26] for detection of follicles. Similarly, [27] used co-occurrence matrix based features for classification of pulmonary nodules and [28] used co-occurrence matrix based color and grayscale texture features for classification of colon cancer images. Local binary patterns (LBP) have been used by [13] as one of the textural features for characterizing stroma development in neuroblastoma and [29] showed the effectiveness of local binary patterns in classification of cell phenotype images as well as for the detection of abnormal pap smear cells. Other applications of LBP in medical image segmentation include the analysis of MRI images [30] and medical image retrieval [31].

While texture based approaches can be used to segment the tissue into relevant structures, often a higher-level representation of the image is required to characterize disease progression or classify it as being malignant or benign. Graph based approaches have been used to characterize tissue malignancy [32], [33]. The use of graphs enables the representation of tissue in a manner that allows mathematical quantification of the spatial arrangements of tissue components. Such representations can provide new insights into the disease. For example, [34] used Voronoi tessellations to represent the distribution of neurons in NeuN stained cortical tissues in order to quantify temporal lobe sclerosis. A model based approach was used by [35] for representing images of follicular lymphoma in terms of the spatial arrangements of the nuclei and cytoplasm.

Along with the advancements in imaging technology, significant advances have also been made in the analysis of disease tissue from a proteomic point of view. A new technology called imaging mass spectrometry (IMS) has emerged as a powerful tool for studying the spatial arrangement of proteins, peptides, lipids and small molecules.
in tissue [36,37]. By providing localized protein profiles in a tissue it may be possible to relate changes in tissue histology to changes in the protein signature of the tissue sample brought about by the change is the number of malignant cells in the tissue. Recent studies have shown that by comparing spectra from normal and diseased tissue, it is possible to determine potential biomarkers that can indicate the presence or absence of disease [38, 39] and may also be used to determine disease progression and patient prognosis [40]. A histology guided approach using MALDI-MS profiling has also been used for proteomic analysis of specific tissues sites [41–43] and for classifying tissue samples into different cancer types.

As imaging technology continues to improve and provide researchers with more and more data, new challenges have emerged - the analysis of whole slide images obtained from these scanners becomes a significant issue to be resolved. Similarly, with images sizes ranging from 30,000 x 30,000 pixels to over 100,000 x 90,000 pixels, the very act of viewing these images on different platforms is a problem. These image resolutions lead to file sizes as large as 30GB in uncompressed form and dissemination of the data to researchers becomes a challenge. Even with modern high speed network access, sharing the original image files across the network is not practical. Thus, there needs to be a more convenient and effective way for sharing this data to foster collaboration, research and education.

1.2 Contributions of this dissertation

The goals of this dissertation are to study the use of computer aided image analysis methods as applied to the analysis of H&E and IHC stained tissue biopsies of follicular lymphoma and lupus. This dissertation is divided into five chapters. Chapters 2 and
Chapter 3 discusses the application of computer-aided image analysis tools for follicular lymphoma and lupus respectively. Chapter 2 focuses on the analysis of IHC-stained tissue of follicular lymphoma for the identification of follicles. We propose the use of color and texture features for identifying salient structures in the tissue. We also discuss the use of a new technology called Matrix Assisted Laser Deposition-Ionization Mass Spectrometry (MALDI-MS) [39, 44] for identifying follicles in tissue. The premise behind this approach is the fact that different types of cells have different protein signatures. Chapter 3 discusses the analysis of mouse and human renal biopsies towards the goal of quantification of IHC stained renal tissue for lupus characterization. Mouse models are widely used [45–47] in lupus research and the availability of whole kidney sections from mice provides a rich dataset for development of image analysis tools for the detection of glomeruli in the kidney.

Chapter 4 focuses on the computational challenges presented by large image data sizes and develops approaches to making analysis of whole slide images practical. We review various approaches to parallelism currently available and discuss the tradeoffs and advantages of each technology. We also discuss and analyze one approach to parallelizing the algorithm for follicle identification developed in Chapter 2. Parallelization of the algorithm introduces some additional problems that are also addressed in this chapter. Finally, we present a performance comparison between the serial and parallel implementation of the algorithm.

Chapter 5 introduces the concept of Virtual Microscopy and focuses on the usability aspects of high resolution images in medical research and education. We discuss the various ways that data can be disseminated to the medical and education community in a manner that can lead to a widespread adoption of this technology. By
developing a web-based approach, the data can now be accessed from a wide variety of platforms ranging from desktop PCs and laptops to smartphones and tablet computers. This type of a technology has the potential to enable pathologists and researchers to leverage computer aided tools more effectively. We discuss the implementation of such a system using the Google Maps technology and the potential adoption of this approach in teaching and research.
Chapter 2: COMPUTER AIDED ANALYSIS OF TISSUE BIOPSIES OF FOLLICULAR LYMPHOMA

2.1 Introduction

Follicular lymphomas (FL) are malignancies of the lymph nodes and are composed of follicle center cells [48], which are a mixture of centrocytes and centroblasts. FL is the second most common non-Hodgkins lymphoma in the United States, accounting for 35% of all adult B cell lymphomas and 70% of low grade lymphomas in U.S. clinical trials. Depending on the grade of the disease, FL may be cured with aggressive chemotherapy. The low grade lymphomas show rapid progression and early identification of such tumors is important for prescribing the appropriate treatment regimen.

2.1.1 Diagnosis and classification of follicular lymphoma

The diagnosis of FL is based on a system adopted by the World Health Organization (WHO) and is based on the average number of centroblasts (CB) in ten random, microscopic high power fields (HPF) [49]. FL can be classified into three grades, with Grade I corresponding to 5 CB/HPF, Grade II with 6-15 CB/HPF and Grade III with more than 15 CB/HPF. Grades I and II are considered indolent with long
average survival rates and patients with these grades of FL and low clinical stage do not require chemotherapy. In contrast, grade III FL is an aggressive disease that is rapidly fatal if not immediately treated with aggressive chemotherapy. As a result of these differences, accurate histological grading of FL is critical for appropriate risk stratification of FL patients and for guiding crucial clinical decisions of timing and type of chemotherapy.

Establishing the diagnosis of FL in the majority of cases is not difficult and is based on characteristic morphologic and immunophenotypic findings using H&E and IHC stained tissue sections. Histological grading of FL using WHO adopted manual method is labor intensive, difficult and shows poor reproducibility. Currently, the inter-reader agreement between pathologists in grading FL is extremely low: in a multi-site study, the agreement among experts for the various grades of follicular lymphoma varied between 61% and 73% [3]. These grade-related differences also underscore the continuous need for a precise and reproducible system for the histological grading of FL.

2.1.2 Methodology

The diagnosis of FL uses both IHC and H&E stained tissues. The H&E stained tissue is used to count the number of centroblasts in follicles. However, it is not always easy to locate the follicle boundaries in the H&E stained tissue. Thus, for the grading of FL, it is important to first identify follicle regions. The use of the CD10 and CD20 stains makes this task significantly easier because of the staining characteristics of these particular stains. Figure 2.1 shows sample follicle regions in CD10 and CD20 stained images. Follicle regions are typically circular to elliptical in shape but they
exhibit a large variation in size. They are presented as dark brown regions separated by lighter brown or blue regions as seen in the figure. This characteristic makes it easy to identify the follicles in low magnifications.

Figure 2.1: Follicles in CD10 and CD20 stained tissues: Images scaled down to 2x microscope resolution

(a) CD10 stained tissue

(b) CD20 stained tissue

Figure 2.1: Follicles in CD10 and CD20 stained tissues: Images scaled down to 2x microscope resolution
An adjacent tissue slice stained with H&E is then used to map the follicles from the IHC image on to the H&E image. Since the H&E image provides details of cellular components [15] such as the nuclei and cytoplasm, the task of counting centroblasts is performed on the H&E image. Figure 2.2 shows adjacent slices of the same tissue stained with IHC and H&E stains.

![Figure 2.2](image)

Comparing the stained tissues in Figure 2.2(a) and 2.2(b) it can be seen that the IHC stained tissue makes it much easier to view the follicle regions as compared with the H&E stained image. Thus, in the development of an automated system for FL grading, we use IHC stained tissue images for follicle identification. This research mainly involves the identification of follicles in adjacent IHC stained tissue sections.
2.2 Follicle detection in IHC images

FL grading is based on the number of centroblasts in ten random standard microscopic high power fields representing malignant follicles in H&E stained tissue. To build an effective computer-aided system for FL diagnosis and grading, it is important to identify the follicle regions in the tissue. IHC stains help identify follicle regions in tissue which can then be registered with the adjacent H&E stained tissue image. The H&E stained tissue can then be used to count the number of centroblasts in the follicles. Figures 2.3(a) and 2.3(b) show two sample images of tissue stained with CD10 and CD20 stains, respectively. Follicle regions are marked with blue boundaries in both images.

From Figure 2.3 we observe that under low magnification, the follicular patterns appear similar in both images. Follicles can range in shape from circular to elliptical with a large variation in the size depending on the tissue. The follicle regions are comprised of brown hues with varying amount of blue regions separating the follicles. Based on the characteristics of the stained tissue images, we propose a novel three step process for the segmentation of the individual follicles in the image. The first step performs a rough segmentation of the follicles using a combination of color and texture features, the second step separates overlapping follicle regions into individual follicles and the third step smoothes the estimated follicle boundaries. The flowchart for the algorithm is shown in Figure 2.4.

2.3 Rough segmentation of follicles using clustering

Rough segmentation of follicles is achieved by clustering the image into four classes. The four classes consist of the slide background, blue cells or nuclei, dark
brown regions that consist of tightly packed cells with membrane staining and light brown regions of the tissue. These classes are identified using a three element feature vector that consists of one color feature and two texture features calculated from the image as described in this section.

2.3.1 Color feature

Color is a very important feature that is used to identify objects of interest in images. Color images are typically represented as linear combinations of the primary colors: Red, Green and Blue (RGB). However, in RGB images, there is significant
correlation between the three color channels and may not be a suitable representation for image processing purposes. Several other color-spaces such as the YIQ, HSV, CIE La*b are defined for representing color images. In our work, we have used the HSV (Hue, Saturation, Value) color-space because it decouples color information from the intensity information and is similar human color perception [50]. The Hue channel describes color, (i.e. red, green, blue, etc.) and can be used to identify colors of interest. In CD10 and CD20 stained tissues, the regions of interest consist mainly of brown hues separated by blue and/or background regions. Therefore, we used the pixel values in the Hue channel as one of the features for clustering. A sample histogram is shown in Figure 2.5.

2.3.2 Smoothing

The second feature is obtained through the reduction in texture variations inside the follicle regions by a smoothing operation on a grayscale version of the RGB image.
Figure 2.5: Sample histogram of Hue channel from CD10 stained image
While a low-pass filter may be used, it has the disadvantage of smoothing edges in the image. We use a median filter for smoothing the textural variations inside follicle while preserving the follicle boundaries. We use a median filter with a large kernel size for smoothing the grayscale image under consideration. For 4x resolution images, a kernel size of 15 x 15 is used. Because the median filtered image demonstrates low contrast, a histogram equalization operation is used to increase the contrast of the image. The equalization operation also helps in better identifying the follicle regions. The pixel values in the histogram-equalized image was then used as the second feature.

2.3.3 Texture energy using co-occurrence matrix

The last feature used for follicle identification is calculated using gray level co-occurrence matrices. Gray level co-occurrence matrices provide a statistical method for characterizing spatial dependencies of gray levels in an image [22]. Co-occurrence matrices measure the frequency of occurrence of pairs of gray values separated by a fixed distance and angle. Using the co-occurrence matrix, a number of textural attributes to characterize properties such homogeneity, contrast, energy, complexity of image, etc can be calculated [22]. In the detection of follicles it is observed that follicle regions consist of tightly packed cells and thus can be seen as having more textural energy than non-follicle regions in the image. Therefore, the energy metric is used as a feature vector for determining follicle locations.

2.3.4 Clustering

The three element feature vector consisting of the Hue channel values, histogram equalized output of the median filter and the energy vector calculated from the co-occurrence matrix were used to cluster the image into follicle and non-follicle regions.
We use the K-means clustering algorithm for this purpose. The K-means algorithm is an unsupervised clustering method for partitioning data into K subsets. In CD10 images, we expect to find four clusters. The clusters consist of the slide background, cells or nuclei that have been stained blue, follicle regions corresponding to the darker brown regions and regions that appear light brown due to sparcity of cells in the tissue.

Figure 2.6: Example scatter plot and labeled image for test image after k-means clustering
After K-means clustering, we identify the cluster corresponding to follicle region by using the average of the median filtered pixel value of all pixels assigned to a given cluster. Since the follicles are expected to be the darkest regions in the tissue, the cluster with the lowest average pixel value corresponds to the follicles in the image and a segmented image is obtained by setting the corresponding pixel locations to 1 and all other pixels to 0. Since this process can produce regions with mis-classified pixels inside follicles, we use morphological filling to enforce a smoothness constraint on the binary image. We apply the watershed transform to the distance transform of this binary image as described in the next section.

2.4 Overlapping follicle separation

The K-means algorithm described in Section 2.3.4 is used to produce a segmented image in which follicle regions are marked as 1 and background regions as 0. However, this process may result in multiple follicles that are merged into one region and must be separated. This is achieved using the watershed transform.

The watershed transform can lead to over-segmentation and several approaches have been proposed for reducing the over-segmentation such as the use of shape priors. However, this typically requires a representative training set. Due to the large variation in follicle shapes, this approach is not suited for our task. Another approach is the use of the H-minima transform [50, 51] which is used to suppress minima in the image that are less than a specified value. To overcome the problem of choosing a value for the H-minima transform that works on all potential objects, we employ an adaptive method similar to [52] in which we apply the watershed transform to the image recursively until the segmentation can no longer proceed. Following the
Figure 2.7: Application of recursive watershed transform to split merged objects

notation in [52], let $H(g_I, h)$ be the H-minima transform for threshold $h$ on the inner distance map $g_I$ of the image $I$ and $N_h$ be the number of objects in the image. The algorithm proceeds as follows.

Figure 2.7 shows the results of successive iterations of the algorithm when it is applied to connected objects. This iterative approach was found to be effective in splitting objects that are merged during segmentation. However, in the case of CD10 images it was observed that a narrowing of objects was not a sufficient condition for splitting objects. For example, even though the watershed process separated multiple objects, they were considered a single follicle by the human expert. On the other
1. Set $h = 1$, $h_{adp} = 0$

2. Calculate $H(g_I, h)$, $N_h$ and $N_{h_{adp}}$

3. While $N_h > N_{h_{adp}}$ do :
   (a) $h_{adp} = h$, Set $h = h + 1$
   (b) Calculate $H(g_I, h)$ and $N_h$

Table 2.1: Recursive watershed algorithm for splitting merged regions

hand, in case of CD20 images, this approach was successful in splitting B-cell regions of interest in almost all cases.

2.5 Follicle boundary smoothing

The output of the watershed transform described in Section 2.4 produces disjoint regions that correspond to the follicle regions. However, the boundaries of the follicles are very noisy and further processing is required to smooth the boundaries. We use Fourier descriptors for smoothing object boundaries. Fourier descriptors are used for representing object boundaries [50] by representing the co-ordinates of each pixel on the boundary of an object as a complex number and calculating the Fourier transform of the complex numbers. Boundary smoothing is achieved by using only a subset of the Fourier coefficients for reconstructing the object boundary. Figure 2.8 shows an example of boundary smoothing.
2.6 Results

Images used in this study were acquired using an Aperio ScanScope XT digitizer and scanned at 40x microscope resolution. At 40x resolution, the CD10 image used in this study was 96,899 x 174,600 pixels in size and was scaled down to 4x resolution for analysis. Our proposed algorithm was applied to 8 images of a CD10 stained tissue. Average image size of the test data was 900 x 1400 pixels. Images of different sizes were chosen due to the fact that follicles can have widely varying sizes and shapes and each image used in the paper was large enough to have several follicles. Results of the automated image analysis were compared with ground truth generated with the help of pathologists.

Figure 2.9 shows the follicle boundaries detected by our algorithm on one of the test images, using immunostaining for CD10. The blue boundaries were marked manually by a trained pathologist and red boundaries are generated by the proposed algorithm.
2.6.1 Measuring segmentation accuracy

The goal of this research is to find follicle regions in IHC images. The follicle regions are then registered with an adjacent H&E stained tissue slice. Thus, the shape of the follicles and their location in the tissue are critical to the process. The proposed algorithm is evaluated by comparing the agreement between the manually segmented follicles and the computer segmented follicles by using a similarity index as defined in [53]. This similarity measure is defined as follows

\[
SI = 2 \cdot \frac{n\{M \cap A\}}{n\{M\} + n\{A\}}
\]  

(2.1)

where, M and A are the objects obtained from manual and automatic segmentation respectively and \( n\{A\} \) is the number of elements in set A.
In some cases, the algorithm splits follicles which are treated as a single follicle by the expert pathologist. In such cases, the similarity score is calculated by considering all the split follicles using the formula:

\[
SI = 2 * \frac{n\{M \cap (A1 \cup A2)\}}{n\{M\} + n\{A1\} + n\{A2\}}
\]

where A1 and A2 are the computer detected objects.

Similarity scores were calculated for each follicle in each image. Table 2.2 shows the average similarity scores for 8 test images. The score listed in the table is the average score of all follicles in a given image.

Table 2.2: Similarity scores for 8 test images at 4x microscope resolution: Overall average score 87.11%

<table>
<thead>
<tr>
<th>Image</th>
<th>Avg. Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91.77</td>
</tr>
<tr>
<td>2</td>
<td>89.68</td>
</tr>
<tr>
<td>3</td>
<td>93.47</td>
</tr>
<tr>
<td>4</td>
<td>93.91</td>
</tr>
<tr>
<td>5</td>
<td>92.04</td>
</tr>
<tr>
<td>6</td>
<td>91.38</td>
</tr>
<tr>
<td>7</td>
<td>84.37</td>
</tr>
<tr>
<td>8</td>
<td>60.29</td>
</tr>
</tbody>
</table>

The average similarity score for Image 8 is low because classification of the image into four classes as described in Section 2.3.4 leads to over-segmentation. By reducing the number of k-means classes to three, a significantly better result is obtained. Additional steps are thus necessary to better identify the number of classes in the image based on the content of the image. One approach to adaptive selection of the number of classes is through the examination of the Hue channel histogram. We have observed that in some cases, the image can be classified into only three classes.
consisting of dark brown, light brown and background/blue regions. In such cases, the Hue histogram shows that the amount of blue in the image is significantly lower than in other cases. An example of the Hue histogram from two such images is shown in Figure 2.10.

Figure 2.10: Hue channel histogram for different image content

The proposed algorithm was applied to a whole slide image of CD10 stained tissue of dimension 9,689x17,460 pixels and takes approximately 1 hour 15 minutes to process a single image. Several additional challenges are posed by the large image
size and additional work is necessary to parallelize the algorithm and run on multiple CPUs in order to make the processing practical for clinical application.

2.6.2 Sensitivity to processing parameters

Our algorithm was found to be most sensitive to the median filter size and the number of classes used for k-means clustering. Using smaller median filter sizes leads to follicles being broken up into smaller objects that need significant processing to merge. On the other hand, larger filter sizes lead to too much merging of regions, especially when applied to analysis of CD20 images. This led to our choice of a 15x15 median filter. As discussed in [26], the number of classes in the image can vary depending on the tissue and an adaptive method for selecting the number of classes is necessary.

2.7 Follicle Identification Using MALDI-MS

While the current diagnosis heavily depends on the review of H&E-stained tissues, additional sources of information such as IHC stained images are frequently needed. In order to better quantify the information inherent in the tissue, we have used a relatively new technology called imaging mass spectrometry (IMS). IMS has emerged as a powerful tool for studying the spatial arrangement of proteins, peptides, lipids, and small molecules in tissues [36,37]. The multichannel detection capability of mass spectrometry (MS) enables the position sensitive analysis of hundreds of different molecules in a single experiment. This is achieved by acquiring mass spectra across a sample at precisely defined geometric coordinates. Post acquisition processing of the data enables ion density maps (images) to be generated for any of the detected...
species where the relative intensity of the ions is displayed based on a color intensity scale [54].

IMS can provide localized protein profiles from a tissue, thus making it possible to relate changes in tissue histology to the changes in the protein signature of the sample. By comparing spectra from normal and diseased tissue, it is possible to determine potential biomarkers that can indicate the presence or absence of disease [38,39] and may also be used to determine disease progression and patient prognosis [40]. A histology guided approach using Matrix Assisted Laser Deposition/Ionization Mass Spectrometry (MALDI-MS) profiling has also been used for proteomic analysis of specific tissues sites [41–43] and for classifying tissue samples into different cancer types. In this section we describe a unique methodology that can potentially improve the accuracy of FL diagnosis by using MS data to classify a tissue into disease relevant morphological sections.

The MALDI-MS data consists of the protein profiles obtained from the tissue sample. Figure 2.11 shows the procedure used to obtain protein profiles from a tissue sample. As shown in Figure 2.11, the tissue sample is coated with matrix and placed into the MS source [36] and the laser beam is used to irradiate each sample spot. The resulting ion signals from 200 to 1000 consecutive shots are averaged across the droplet surface in order to generate the mass spectrum. The resulting mass spectra yield signals of various intensities in a m/z range from 2000 up to m/z 200,000 [36,55]. This process is called profiling. Data is collected in a Cartesian grid over the tissue sample. A detailed description of this process can be found in the paper *Imaging mass spectrometry: principles and potentials* by Chaurand, P, et al. (Toxicologic Pathology, Vol. 33, pp. 92-101)
Figure 2.11: MALDI-MS data acquisition procedure
2.8 MALDI-MS data collection

Preliminary data was obtained for a whole-slide FL case. Formalin fixed paraffin embedded (FFPE) tissue sections from healthy tonsil biopsies were used to generate image data as well as MS data because of the clear demarcations of the mantle zones in these images.

Serial 5 µm thick sections were cut from the FFPE tonsil tissue blocks using a microtome. Sections from the tissue blocks were either mounted onto ITO-coated conductive slides for MALDI MS analysis, or onto standard glass microscope slides for H&E staining. Deparafinization of the tissue sections was carried out using xylene and graded ethanol washes. Once the slides were fully dry, a trypsin solution was automatically spotted onto the tissue section using a Portrait 630 reagent multi-spotter (Labcyte, Sunnyvale, CA) into an array incorporating 250 µm center to center spacing between individual spots, each of which were approximately 175 µm in diameter. The trypsin was spotted for over 30 iterations while allowing the trypsin solution to dry following each droplet application. An optically scanned image of the spotted tissue was generated for registration with the H&E stained tissue image. The spotted tissue image was used to map spatial locations of the spots in the tissue to the corresponding mass spectra. The spotted tissue image is relatively low resolution compared with the H&E image.

An H&E stained section of the FFPE tissue section was scanned to generate a high-resolution image of pixel size 59363 x 58311. This high-resolution image was scaled down by a factor of 15 for use in registration. Figure 2.12(a) shows a lower-resolution version of the H&E stained tissue image.
Following trypsin/matrix application, FFPE tonsil tissues were analyzed using an Ultraflex II MALDI TOF/TOF mass spectrometer (Bruker Daltonics, Billerica, MA) controlled by the Flex Control 3.0 software package. The mass spectrometer was operated with positive polarity in reflectron mode and spectra acquired in the range of m/z 700-5000. Image acquisition of the spotted arrays was carried out using the Flex Imaging 2.0 (Bruker Daltonics, Billerica, MA) software package. A total of 1600 spectra were acquired at each spot position in a customized spiral raster pattern in 200 shot increments at a laser frequency of 200 Hz. The customized raster pattern was used to sample the entire spot area. The red spots on the tissue shown in Figure 2.12(b) indicate the locations where mass spectra were acquired. This data is critical to the analysis because it correlates spatial locations on the tissue section with a unique mass spectrum.
2.9 Image analysis

As shown in Figure 2.12(b), the red locations indicating the laser spots are easily identifiable on the spotted tissue image. However, the image also contains non-red spots and the RGB colorspace is not optimal for locating the red spots. The spotted tissue image was first converted to the La*b colorspace. The red spots in the RGB colorspace are converted to bright spots in the a* channel, while the gray-blue spots in the RGB colorspace are observed to have significantly lower values in the same channel. The a* channel was segmented into background and laser spots by thresholding, with the threshold determined using the Otsu method [1]. Image segmentation was followed by a simple edge detection operation. Laser spots were detected by labeling connected components and determining their centroid. Once the laser spots were identified, the locations of the laser spots in the image were mapped to the files containing the MS data.

For the preliminary study, the scaled down H&E image and the laser grid image were registered manually. A non-reflective similarity transform was used after selecting appropriate control points in the two images. The registration process was completed using the Image Processing Toolbox in MATLAB (Mathworks, Natick, MA). The two images were registered after dividing them into four separate sub-images in order to localize the effects of distortion caused during the sectioning process. The locations of laser spots were transferred to the H&E stained image as shown in Figure 2.14(a). An expert hematopathologist reviewed these locations and established the ground truth for the subsequent analysis.
Figure 2.13: Identifying laser spot location on MALDI-MS grid: Image segmentation using Otsu [1] thresholding, edge detection and image morphology
2.10 MALDI-MS data analysis

In order to characterize different sections of the tissue that are morphologically relevant to disease state, it was first necessary to identify laser spot locations in the mantle, follicle and intra-follicular regions of the tissue as shown in Figure 2.14(b). The red location corresponds to the mantle zone, green corresponds to the follicle and blue corresponds to the intra-follicular region. Figure 2.15 shows a sample spectrum obtained from a mantle zone location. Data points that were not located in the mantle, follicle or intra-follicle regions were not used in the analysis because they are not considered relevant to the disease state.
Figure 2.15: Sample mass spectrum plots from representative mantle zone and follicle regions.
Table 2.3: Confusion matrix as a percentage for leave one out validation using SVM classifier

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Mantle</th>
<th>Follicle</th>
<th>Intra-Follicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantle</td>
<td>40.00</td>
<td>4.00</td>
<td>56.00</td>
</tr>
<tr>
<td>Follicle</td>
<td>23.52</td>
<td>64.70</td>
<td>11.76</td>
</tr>
<tr>
<td>Intra-Follicle</td>
<td>4.46</td>
<td>2.23</td>
<td>93.29</td>
</tr>
</tbody>
</table>

2.10.1 Methodology

MS data from the selected points of interest was first baseline corrected for noise and background reduction and then normalized. Previous studies have compared mass spectra from normal and disease tissue to identify significant bio-markers. Since a single normal tissue sample was used in the study, significant bio-markers could not be identified by simple comparison of protein profiles from different tissue sections. Therefore, the complete spectra consisting of 56,492 data points each were used to develop classifiers to distinguish between the three tissue regions. Common statistical techniques applied for the analysis and classification of mass spectrometry data are Principle Component Analysis (PCA), Support Vector Machines (SVM) and K-Nearest Neighbor classification [38, 41, 56, 57].

In this study we used the Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) classifiers available in the Statistics Toolbox for MATLAB. A leave-one-out cross validation study was performed using the SVM and KNN classifiers. The results are summarized in Table 2.3 and 2.4.

A second analysis similar to the procedure described by Ressom et al in [38] was used to perform a K-fold cross validation study. The availability of data points
Table 2.4: Confusion matrix as a percentage for leave one out validation using KNN classifier

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Mantle</th>
<th>Follicle</th>
<th>Intra-Follicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantle</td>
<td>40.00</td>
<td>24.00</td>
<td>36.00</td>
</tr>
<tr>
<td>Follicle</td>
<td>47.05</td>
<td>52.94</td>
<td>0.00</td>
</tr>
<tr>
<td>Intra-Follicle</td>
<td>11.17</td>
<td>4.46</td>
<td>84.35</td>
</tr>
</tbody>
</table>

from each class used in the analysis is dictated by the physical characteristics of the tissue being studied. As a result, in this study a total of 189 spectra from the intra-follicular region were available as compared with 31 for the follicle regions and 43 for the mantle zone. To avoid biasing the classifiers, only 43 spectra from the intra-follicle region were used along with all the data from mantle and follicle regions. Tables 2.5 and 2.6 summarize the results of this analysis averaged over 200 runs of the K-fold cross validation procedure. In each run, training and testing sets were determined randomly, however, the total number of spectra from each class were the same. It is seen that both classifiers perform well in identifying the intra-follicle and mantle regions. However, the follicle regions are misclassified into the mantle zone almost 50% of the time.

2.11 Discussion

Tables 2.3, 2.4, 2.5 and 2.6 summarize the classification accuracy of the SVM and KNN methods. A potential cause of the lower classification accuracy in the case of mantle vs. follicle is that in several cases, the mass spectrometry data was obtained from a region overlapping both the mantle zone and the follicle region as
Table 2.5: Confusion matrix as a percentage for leave one out validation using SVM classifier

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Computer</th>
<th>Mantle</th>
<th>Follicle</th>
<th>Intra-Follicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantle</td>
<td></td>
<td>85.55</td>
<td>9.55</td>
<td>4.90</td>
</tr>
<tr>
<td>Follicle</td>
<td></td>
<td>41.73</td>
<td>53.45</td>
<td>4.82</td>
</tr>
<tr>
<td>Intra-Follicle</td>
<td></td>
<td>22.98</td>
<td>0.36</td>
<td>76.65</td>
</tr>
</tbody>
</table>

Table 2.6: Confusion matrix as a percentage for leave one out validation using KNN classifier

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Computer</th>
<th>Mantle</th>
<th>Follicle</th>
<th>Intra-Follicle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantle</td>
<td></td>
<td>48.27</td>
<td>42.22</td>
<td>9.51</td>
</tr>
<tr>
<td>Follicle</td>
<td></td>
<td>45.56</td>
<td>50.98</td>
<td>3.46</td>
</tr>
<tr>
<td>Intra-Follicle</td>
<td></td>
<td>20.22</td>
<td>4.26</td>
<td>75.52</td>
</tr>
</tbody>
</table>

Figure 2.16: Illustration of challenges to MALDI-MS analysis: Laser spots from MALDI-MS acquisition may include spectra from multiple different regions
shown in Figure 2.16. In such cases, the data is assigned by the human expert to either one of the regions based on the location of a majority of the laser spot inside either one of the regions. This can affect the results of the classifier since the mass spectra now contains contributions from two regions of tissue. In Figure 2.16 two regions classified as mantle (red) and follicle (green) respectively show this problem. The inability of the classifiers to distinguish between the mantle and follicle regions with high rates of accuracy can also be ascribed to the fact that these two regions of tissue consist of similar types of cells. Therefore, it is reasonable to expect that the protein profiles of these regions are similar in nature. Another possible cause of this error is the inaccuracy in the registration of the high resolution H&E image and the MADI-MS matrix image. Because of the disparity in the resolutions of the two images, the high resolution image must be downsampled or the low resolution image must be upsampled. The scaling factor in this case can be significantly large since the H&E image is scanned at 40x microscope resolution, which is not the case for the MALDI-MS matrix image.

In this study, all data points in each mass spectrum were considered in the analysis which makes the problem computationally expensive. However, the results from this pilot study may be used to identify significant markers that can be used for reducing the data dimensionality, thus allowing the problem to be reduced to a more tractable size.
Chapter 3: SEMI-AUTOMATED QUANTIFICATION OF IHC AND H&E STAINED RENAL BIOPSIES OF LUPUS

Systemic lupus erythematosus (SLE) involvement of the kidneys is associated with high morbidity and mortality. Despite the advances in treatment of lupus nephritis (LN), significant variation in response to treatment exists. While some of this variation is undoubtedly related to genetic and environmental factors, inter-observer variation and sampling bias continue to confound the accurate interpretation of kidney biopsies, contributing to variations in treatment response [58]. Underestimating the severity of renal involvement or inaccurately assessing reversibility of lesions can result in significant morbidities related to under or over-treatment. Currently, sampling error is reduced by increasing the amount of tissue obtained [59–61], but this approach is limited due to safety concerns. Inter-observer variation is reduced by following standardized protocols in processing kidney biopsy specimens and scoring them. However, this examination is tedious, time consuming, error-prone, and at best only semi-quantitative. Therefore, the development of new, more objective tools that can assist in the process of renal biopsy interpretation in LN is warranted.

This chapter discusses two main approaches being investigated - Section 3.1 discusses the quantification of positively stained cells in immunohistochemical (IHC) stained human tissue biopsies. This approach assumes that locations of glomeruli in
the biopsy are either known or manually identified. Section 3.4 discusses the development of a system to identify glomeruli in H&E stained mouse renal biopsies, which can be potentially applied to human biopsies.

3.1 Quantification of IHC Staining in Images of Renal Biopsies

Immunohistochemistry is currently not an integral part of a complete clinical examination of a kidney biopsy. However, understanding the types of infiltrating leukocytes may lead to a better understanding of the pathogenesis of kidney injury in LN, and could lead to identifying potential new therapeutic targets. Immunohistochemistry can easily differentially identify leukocytes subsets, both broadly in terms of lymphocytes, monocytes/macrophages, and neutrophils, and more finely, in terms of regulatory and effector T-cells and macrophages. However, quantification of these subsets, and precise localization with respect to other kidney structures by manual counting and planar observation are prone to errors. The use of computer-aided image analysis tools can provide an accurate distribution and count of cell types and assist in the localization of inflammatory infiltrates.

Figure 3.1 shows selected regions at 40x microscope resolution from two different renal tissues stained for the CD3 antibody. It can be seen that even with the same antibody stain, the procedure used can significantly affect the overall color characteristics of the tissue. In the CD3 stained tissue, the goal is to identify cellular infiltrates that are identified as the dark brown stained cells in the tissue. In general, this is true for most IHC stains, i.e., the cells of interest are typically stained dark brown and the rest of the tissue is stained in shades of blue. Thus, in Figures 3.1(a) and 3.1(b), the first goal is to identify the cells that are stained dark brown. Once these
cells have been identified, we aim to find the distribution of the CD3-positive cells throughout the tissue with respect to chosen landmarks such as the glomeruli and the tubules in the sample.
Figure 3.2: Proposed algorithm for quantification of IHC stained tissue images of Lupus

3.2 Proposed Approach

We propose a color based segmentation approach, followed by standard morphological post-processing techniques for the semi-automated quantification of IHC-stained tissue images of lupus. The steps in the proposed algorithm are as follows:

1. Color-space conversion of images from RGB to CMYK

2. Thresholding of the Yellow channel in the CMYK colorspace

3. Morphological cleanup including recursive watershed
4. Enumeration of detected cells

Figure 3.2 shows the different steps in this process for a sample image.

3.2.1 Color Segmentation

Typically, the perceptually uniform La\textsuperscript{*}b or the HSV color spaces are used for color based segmentation because of the advantages offered by the respective color models in differentiating colors or closely modeling the perception of colors by the human visual system. However, there exists another colorspace, called the CMYK colorspace, which is typically used in the print industry [62]. The CMYK space is a subtractive colorspace, i.e. colors are produced by subtracting specific spectra [63] and uses Cyan, Magenta, Yellow and Black as the primary pigment colors. Recent work by [64] has shown that the CMYK color model can effectively be used for analysis of IHC stained tissue images. Pham et al in [64] show that the Yellow channel in the CMYK colorspace achieves high contrast between different chromogen intensities and is also independent of observer bias. Similarly, [65] used the CMYK colorspace for assessment of immunohistochemically stained tissues of cervical cancer and intraepithelial lesions. Because of the evidence of successful use of the CMYK colorspace for analyzing IHC stained tissue images, we propose the use of this colorspace for the quantification of IHC stained renal biopsies.

Applied to images of the IHC stained tissue images shown in Figure 3.1, we observe that using the Yellow channel in CMYK colorspace serves to highlight the CD3-positive cells in tissue very effectively. Figures 3.3 and 3.4 show the Cyan, Magenta, Yellow and Black channels of the images shown in Figure 3.1. In the case of both these images, the yellow channel servers best to highlight the CD2-positive cells. This
Figure 3.3: CMYK color conversion: Image from Fig. 3.1(a) converted to the CMYK colorspace

can be seen more clearly in Figure 3.5 where the Yellow channel for both images is shown.

As seen in Figures 3.5(c) & 3.5(d), the Yellow channel tends to have a predominant peak in the lower half of the histogram, corresponding to regions of the tissue that are stained blue or light brown. The upper half of the histogram corresponds to the regions of tissue with CD2-positive cells. We can use the Otsu [1] method to threshold the Yellow channel such that the segmented image provides the location of the CD3-positive cells. The result of using Ostu segmentation are shown in Figure 3.6.
3.2.2 Post Processing Segmentation Results

While the Otsu segmentation results in identifying the regions of interest in the tissue, additional post-processing steps are necessary. Specifically, there are two problems that must be resolved. These include the removal of small regions that do not correspond to CD3-positive cells and splitting merged cells.

The problem of removing objects that do not correspond to regions of interest can be resolved by using morphological operations. The first step is to use morphological opening to remove regions that are smaller than the expected nuclei. In this case, all areas of tissue with fewer than 100 pixels. The result of such an operation can be seen in Figures 3.6(b) and 3.6(d). The second problem of splitting merged cells can be overcome through the use of an iterative watershed process for splitting merged objects as described in Section 2.4.
Figure 3.5: Yellow channel in CMYK: Tissues shown in Fig. 3.1 were converted to the CMYK colorspace. The yellow channel for both images and the corresponding histograms are shown in this figure.
Figure 3.6: Segmentation results: Otsu segmentation can be used to identify CD3-positive cells in the tissue

3.2.3 Quantification of CD3 Staining

The primary goal of analyzing CD3 stained tissue is to quantify the amount of CD3-positive staining in a tissue. Towards this goal, the first step was to segment the tissue image such that all CD3-positive cells are identified. Once the cells have been segmented as described in the previous section, the next step is to calculate the distribution of the cells across the tissue and in relation to significant landmarks in the tissue. Because Lupus affects the functioning of the glomeruli, the first landmark to be chosen are the glomeruli in the tissue. Secondly, tubules in the kidney are chosen as the other landmark for quantification.
Figure 3.7: Quantification of CD3 positive cell distribution: Concentric circles of increasing radius around selected locations in the tissue are used for quantifying the distribution of CD3 positive cells across the tissue.

Quantification is carried out by first identifying glomeruli border manually. We then construct concentric circles around the centroid of the glomeruli with radii increasing in 100 µm increments. Thus, we can identify annular rings of 100 µm thickness around each glomeruli. The maximum radius used for the concentric circles is 600 µm. This choice is dictated by the small amount of tissue available in a human kidney.
Figure 3.8: Example of positive stained cell detection around selected glomeruli in two different tissues: Cells in region of interest are shown in green.

Positively stained cells are then counted in each of the annular rings. While selecting glomeruli to quantify in this manner, care must be taken to ensure that the concentric circles around the glomeruli do not overlap with those around any other glomeruli (or tubule) in the tissue. This ensures that cells are only counted once and are associated with only one glomeruli (or tubule) in the tissue. A similar procedure is used for a few selected tubule locations in each tissue.

Figure 3.8 and 3.9 show four examples of this procedure. In these figures, the regions outlined in yellow correspond to positively stained cells that lie outside the annular ring around the glomeruli (or tubule) of interest. The images shown in these figures were extracted from different tissue samples.
(a) Annular ring of radius 100\(\mu\)m around selected tubule contains 61 cells
(b) Annular ring of radius 100\(\mu\)m around selected tubule contains 15 cells

Figure 3.9: Example of positive stained cell detection around selected tubules in two different tissues: Cells in region of interest are shown in green

3.3 Results

We used seven renal biopsies stained with the CD3 stain. Glomeruli in each tissue were selected so that there was no overlap of the annular regions corresponding to different glomeruli. Similarly, at least one tubule location was chosen in each tissue image such that the concentric rings around the tubule did not overlay with any of the concentric rings around the chosen glomeruli in the same tissue. As illustrated in Figure 3.7(a), the green and red circles have no overlap. The quantification procedure described in Section 3.2.3 was applied to the images and the number of positively stained cells in concentric rings around the selected glomeruli and tubules were counted. The results of this procedure are shown in Figures 3.10 and 3.11. We
Figure 3.10: Quantification of infiltrate around glomeruli: Number of positive cells found in concentric rings of 100µm around glomeruli in each tissue. Unique colors in each plot signify a different glomerulus in the tissue.

We observe an overall trend of increasing number of cells around the glomeruli as we move away from the glomeruli center. A similar trend is also seen around the tubules.

While it is possible to quantify CD3 stained renal tissue in this manner, one of the main problems that we encounter is the fact that imposing a non-overlapping criteria on the concentric circles around the glomeruli and tubules leads to the discarding of several glomeruli in a tissue. Similarly, the regions of tissue from which tubules can be selected without overlapping the circles is also severely restricted. Thus, while
Figure 3.11: Quantification of infiltrate around tubules: Number of positive cells found in concentric rings of 100µm around selected tubules in each tissue. Unique colors in each plot signify a different tubule.
these results show the quantification in a tissue, alternative means of quantification and localization of the cell counts is necessary to obtain more accurate and relevant results. One of the ways this might be done is to select all glomeruli and a few tubule locations in each tissue without considering the overlap between the concentric circles. For example, consider the tissue order depicted in black in Figure 3.12. The red regions outline glomeruli in the tissue and blue regions outline the selected tubule locations. We use the Watershed transform to separate these components such that each object now has a unique polygonal area associated with it. We can then perform quantification of tissue infiltrates in this area for each object thus selected. The green lines in the figure show the boundaries of these polygons. Quantification of infiltrates in this manner is not easily done manually and may give new insights into the disease.
Figure 3.12: Alternative method for quantification of IHC-stained tissue images: Red regions depict glomeruli locations, blue indicates selected tubule locations and green lines indicate regions around a glomeruli or tubule to be associated with the specific object.
3.4 Glomeruli Detection in H&E Stained Mouse Renal Biopsies

Renal biopsies from SLE patients exhibit a spectrum of vascular, glomerular and tubulointerstitial lesions [66]. Our goal in this work is to develop automated image analysis techniques to identify glomeruli in kidney biopsies and to quantify characteristics of the glomeruli. Typically, human kidney biopsies contain only a few glomeruli since a minimal amount of tissue is extracted. This presents a significant challenge since the amount of data available for development of image analysis algorithms is limited. Mouse models have been used extensively for understanding the pathogenesis of SLE [45] and offer a potential solution to the problem of insufficient, representative data. Past studies have shown that an entire mouse kidney may have numbers of glomeruli ranging from 2,645 to over 15,000 [46], [47]. With the use of murine models, it is possible to obtain biopsies of the entire kidney which results in the availability of a significantly large number of glomeruli in a given serial section. Therefore, we have focused our research on the identification of glomeruli in H&E stained mouse renal biopsies.

Glomeruli consist of small knots of capillaries and supporting structures that are suspended within structures known as Bowman’s capsules. Examples of glomeruli in mouse renal biopsies stained with H&E are shown in Figure 3.14. Glomeruli in the tissue images shown here are outlined in green. As seen in Figures 3.14(a) and 3.14(b), the glomeruli are identified primarily by the absence of tissue, which in turn is manifested as regions of the tissue that are not stained by the H&E stain. The region around the glomeruli is known as the Bowman’s space and is shown in Figure
Figure 3.13: Mouse Glomerulus: Schematic diagram and a sample glomerulus section at 40x resolution. Schematic from [2]

3.13(b), highlighted in blue. In this research, we use the presence of the Bowman’s space as the marker for glomeruli identification.

The challenges in developing robust image analysis methods include wide variation in the image quality caused by differences in staining protocols used, age of the slides and non-uniformity of tissue sections due to fixation. As seen in Figures 3.14(a) and 3.14(b), the same stain can produce vastly different images. Clearly, using the RGB space for analysis is not ideal in such cases. Another challenge in the identification of glomeruli in such tissue images is the wide variation in overall shape of the glomeruli. While glomeruli are expected to be circular in shape, a large amount of variation in size is observed. This can be caused by the disease state of the mouse as well as the distortions introduced during the sectioning and staining process. Typically, mouse glomeruli are 70 µm in diameter and this fact is used in our proposed approach for glomeruli segmentation.
3.4.1 Proposed Algorithm for Glomeruli Segmentation

We propose an algorithm for glomeruli detection that uses color features for segmentation of the tissue to identify all potential regions of interest in the image. After the initial segmentation, we remove all objects that are too large or too small to be sections of Bowman’s spaces surrounding the glomeruli in the tissue. This is achieved through the use of morphological operations such as object removal and erosion. An additional cleanup step employed is the removal of all objects for whom the centroid lies on the object itself. This step results in the removal of circular objects or large objects that result from tears in the tissue. This cleanup process results in the removal of most of the segmented objects that are considered noise for our task.

The next step in the proposed algorithm is the task of grouping segmented objects such that all objects around the same glomerulus are associated with each other. This process uses rules of perceptual grouping and employs characteristics of the objects such as the directionality, orientation and curvature to perform the grouping.
Figure 3.15: Proposed algorithm for glomeruli detection in H&E stained mouse renal biopsies
The proposed steps for identification of glomeruli in H&E stained mouse renal biopsies is shown in Figure 3.15 and can be summarized as follows:

1. Color based segmentation for identification all possible regions corresponding to Bowman’s space
2. Morphological cleanup for noise removal
3. Perceptual grouping of objects to identify glomeruli

3.4.2 Color Based Segmentation

For the color based segmentation of H&E stained tissue images, we propose using a combination of the $La^*b$ and the CMYK colorspace. Specifically, we use $a^*$ channel from the $La^*b$ colorspace and the magenta channel from the CMYK colorspace as the feature vectors for thresholding the image using k-means clustering.

Figure 3.16(a) shows the magenta channel and 3.16(b) shows the $a^*$ channel for a region of interest extracted from one of the renal biopsies used in this study. As seen in both figures, the Bowman’s space is manifested as a dark region in these color channels. Using these two color channels as the feature vectors in a k-means step, all pixels in the image are clustered into four classes. The Bowman’s gap is identified as the set of pixels with the lowest average value in the magenta channel. This is followed by morphological cleanup to remove noise. The cleaned, thresholded image is shown in Figure 3.16(c) and the borders are outlined in blue on the original image in Figure 3.16(d).

This approach was applied to the segmentation of several small regions of interest extracted at 40x microscope resolution from four different tissue biopsies. The results of this processing are shown in Figure 3.18 and 3.19. It can be seen that the proposed
Figure 3.16: Color-based segmentation of glomeruli in H&E stained tissue

The approach is well suited for identifying potential regions corresponding to Bowman’s space in the tissue and in turn the glomeruli location. Although this approach has been found to be successful, there are two main problems that have to be resolved. The first problem is the fact that the appearance of Bowman’s spaces is indistinguishable from absence of tissue due to breaks in the tissue which leads to the slide background showing through the sample. This can result in regions that are significantly larger than a Bowman space or can be too thick to be considered a Bowman’s space. We can use this fact to remove most of these outliers. Additionally, we remove all objects that are simply too small to be significant. In this research, we remove all objects that have less than 400 pixels.

The second problem is a more difficult problem to resolve and involves the organization of the objects observed in the segmented image to determine all fragments that
Figure 3.17: Morphological cleanup: Removing small objects and non-curved objects in the thresholded image

are part of the Bowman’s space around the same glomerulus. The segmentation process has no knowledge of the morphology or spatial arrangement of the glomeruli and the Bowman’s space surrounding it. While a large number of objects in the thresholded image correspond to Bowman’s space, additional post-processing is necessary
to group only those objects that actually correspond to the Bowman’s space. We propose a methodology of grouping the segmented regions that relies on perceptual grouping of objects.

### 3.4.3 Curve extraction

Perceptual grouping is typically carried out on images that comprise of edges/segments that are 1-pixel in thickness. The output of our segmentation step results in
Figure 3.20: Segmented objects and the desired curve representation of the objects shown in red

objects of varying thickness and shapes as shown in Figure 3.16(c). This image is first converted into line segments of 1-pixel thickness such that the segments are a true representation of the objects in the image. For example, consider two segmented objects shown in Figure 3.20(a) and the desired representations of the two objects shown in red in Figure 3.20(b). The curves representing the two objects capture physical properties that are most important for our task: curvature, orientation, directionality. These properties are discussed in more detail in Section 3.4.5.

One way of converting these images to line drawings is to skeletonize all the objects in the image. The two objects shown in Figure 3.20(a) can be segmented using classical morphology approaches as proposed by Soille [51]. The result of such a skeletonization is shown in Figure 3.21(a). However, it can be seen that this skeletonization produces a representation of the original object that can be considered very noisy for our purposes. Here, the noise in the skeleton refers to the abundance of small segments of the skeleton that branch away from what could be considered as
Figure 3.21: Curve extraction from objects: Segmented objects can be skeletonized to obtain representation in the form of 1-pixel thick curves.

Although the fast marching based skeletonization approach gives us a better representation of the original objects than classical morphology, in most instances the resulting skeleton will have several branches, resulting in a skeletonization that has several branches. As shown in Figure 3.20(b), the desired representation of an object...
Figure 3.22: Effect of non-smooth boundaries on the skeletonization of objects: Noise in object boundaries can lead to multiple branches in the object skeleton as illustrated here.

should have exactly two end-points which can then be used to compute the various Gestalt properties of pairs of objects. Multiple branches in a skeleton arise from the fact that the objects themselves are not perfectly smooth. Thus, consider the object shown in Figure 3.22 and the skeletons produced by classical morphology and the approach proposed by [67].

The object shown in Figure 3.22(a) has a border with several protrusions as well as indentations as highlighted in the figure. If these imperfections in the border are removed, the skeleton obtained is smoother and satisfies our requirement that the skeleton have no branches and exactly two end-points. One approach to removing these imperfection is to smooth the object. There are several methods available for smoothing objects. For example, one may use Fourier Descriptors for smoothing by reconstructing the object with just a few Fourier co-efficients. This approach, however, requires the determination of the minimum number of coefficients to be used for a satisfactorily smooth representation of the object, as shown in Figure 3.23. If too
few coefficients are used, the shape of the object is significantly distorted, as shown in Figure 3.23(a). For the object in Figure 3.22(a), using 7 coefficients appears to produce sufficiently smooth objects as a result, but the overall shape is distorted from the original. However, as object sizes and shapes change, the number of coefficients have to be chosen adaptively since one value will not work across all shapes. Another approach to smoothing might be to recursively filter the image with a large median filter. The results of such a process are shown in Figure 3.23(c). This approach also suffers from the same problem of determining optimal values for the filter size and the number of iterations to be used for each object.
Thus, a different approach is needed that preserves the overall shape of the object under consideration while at the same time minimizing the amount of distortion in the border caused by protrusions or indentations in the object.

### 3.4.4 Non-Convex Hull for Object Representation

Our goal is to represent the objects identified from the segmentation process with smooth curves that can be used to generate features that are perceptually significant. Towards this goal, we need a representation of the objects that retains its shape, orientation, size and curvature. In our images, the Bowman’s space around the glomeruli are typically curved and noticeably non-convex. This is an important property of the tissue that we can exploit. Duckham, et al in [69] proposed an algorithm for representing “characteristic” shapes of a set of points in a 2D plane. Given a set of points $P$, the algorithm produces a non-convex polygon containing all points $P$ which is itself contained within the convex hull of the points. The authors refer to this shape as “characteristic shapes” or $\chi$ shapes. We will use the same terminology here. Since these shapes can be non-convex in nature, we also refer to them as non-convex hulls to contrast with convex hulls. The $\chi$ shape of an object is not unique and it can be parameterized by a normalized length parameter $l$ which can be used to generate a number of characteristic shapes for a given set of points. Consider the set of points in Figure 3.24(a). We can generate several different $\chi$ shapes as shown in Figures 3.24(b)-3.24(d). Each of these are valid $\chi$ shapes and the “correct” answer depends on the observer and the question being posed. Thus, the shape in Figure 3.24(d) may be considered the optimal shape if the goal is to identify the C-shape in the figure.
Figure 3.24: Illustration of different characteristic shapes generated from a set of points $P$
Given the set of points $P$, the algorithm for finding the $\chi$ shape of a set of points proceeds as follows [69]:

1. Generate the Delaunay Triangulation of the points $P$

2. Remove the longest exterior edge $E$ of the triangulation such that
   - $\text{length}(E) > l$
   - The exterior edges of the resulting triangulation form a simple polygon

3. Repeat Step 2 until there are edges left to remove or the condition is met

While the length parameter $l$ can take any positive value, the authors in [69] propose normalizing this parameter based on the maximum and minimum edge lengths of the Delaunay Triangulation. If $\max_P$ and $\min_P$ are the maximum and minimum edge lengths the normalized length parameter is $\lambda_P$ is given by:

$$
\lambda_P = \begin{cases} 
1 & \text{if } l \geq \max_P, \\
\frac{l - \max_P}{\max_P - \min_P} & \text{if } \min_P \leq l < \max_P, \\
0 & \text{if } l < \min_P 
\end{cases} \quad (3.1)
$$

Figure 3.25 shows four steps in this algorithm for the set of points $P$ in Figure 3.24(a) for $\lambda_P = 0.5$.

We can now apply this characteristic hull algorithm to the objects identified from our segmentation process to reduce the noise in the object boundaries. Given a segmented object as shown in Figure 3.22(a), can generate different characteristic shapes for the object by changing the parameter $\lambda$, as shown in Figure 3.26.

It can be seen from this figure that the characteristic shape or the non-convex hull of the the object can give us a resulting object that has smoother boundaries while retaining the overall shape of the original object. However, the final result is
Figure 3.25: Calculation of non-convex hull for $\lambda_P = 0.5$: Red indicates the $\chi$ shape at each step and the dashed lines represent the removed edges of the triangulation.
dependent on the choice of \( \lambda \). Since we are interested in generating object skeletons that have minimum number of branches, we propose the automatic selection of \( \lambda \) on an object-by-object basis as follows:

1. Initialize \( \lambda = 0.9 \)

2. At iteration \( N \), determine the \( \chi \) shape of the object

3. Find the skeleton of the \( \chi \) shape and determine the number of branches \( B_N \)

4. If \( B_N = B_{N-1} \) stop, otherwise repeat Steps 2 and 3

This procedure gives us a resulting shape that can be considered optimal for our task. Applying this procedure, the optimal \( \chi \) shape obtained for this object is shown in Figure 3.27(a). We use the resulting \( \text{chi} \) shape to calculate the skeleton of the object and the results are shown in Figures 3.27(b) and 3.27(c).

We apply this procedure to find the skeletons of all objects in the images being analyzed to find the Bowman’s space around glomeruli. Once the objects have been skeletonized and reduced to curves of 1-pixel thickness, we can apply the concepts of perceptual grouping to associate the objects found in the image.
3.4.5 Perceptual Organization

The segmentation procedure proposed in Section 3.4.2 does not take into account the spatial arrangement of objects in the image. Thus, a post-processing step that uses rules of visual perception or “grouping” is proposed as the final step in glomeruli identification. Basic grouping laws that have been proposed include vicinity, similarity, continuity, closure and common motion [70]. These Gestalt laws can be used to find a subset of objects (typically edges/lines/curves obtained from edge detection) that are grouped together in a perceptually salient manner. For example, Wang, et al in [71] use boundary closures, proximity and continuity to develop a graph theoretic approach for identifying salient boundaries in an image. In [72], the authors use saliency measures derived from contour length and circularity measures derived from open arcs to identify salient curves in solar images.

For the purpose of glomeruli identification, the most important visual characteristics are boundary length, proximity, directionality and co-circularity. The boundary length can be used to eliminate any curves that may correspond to noise such as tears.
in the tissue or background regions that are too large to be sections of the Bowman’s space surrounding a glomerulus. The proximity principle can be used to incorporate physical characteristics of the tissue being analyzed. Specifically, we use the fact that mouse glomeruli are typically expected to be 70 µm in diameter. Thus, any segments that are farther than this distance will not be grouped together. Because we are dealing with soft tissue images, using 70 µm as the cut-off distance may not be a sufficient condition in all cases. This is because the sectioning and staining process can cause stretching of the tissue, which in turn can cause the glomeruli to be distorted into a more elliptical shape. In practice, we use 80 µm as the cut-off distance to allow for variations between samples as well as any stretching/deformation of tissue that may have occurred during the fixation process.

![Figure 3.28: Co-circularity derived from segments in an image](image)

The next Gestalt property that we use is the co-circularity metric. The co-circularity metric can be defined as mirror symmetry of the edges being examined [73]. This is illustrated in Figure 3.28. Let $S_i$ and $S_j$ be the two line segments shown in
Figure 3.29: This figure illustrates one of the difficulties with using tangents to the end points of the curve.

Blue in Figure 3.28 and $l_{ij}$ be the line segment connecting the nearest points of the two segments. We construct tangents to the segments at the respective endpoints and calculate angles $\theta_{ij}$ and $\theta_{ji}$ between the segment $l_{ij}$ and the two tangents respectively. Then, the co-circularity metric is calculated as: $\theta_{ij} - \theta_{ji}$. This difference approaches zero when two segments are part of the same circle.

In principle, the end points of the segments are used to construct tangents to the segments under consideration and to find the angles $\theta_1$ and $\theta_2$ as shown in Figure 3.28. However, consider the pair of segments $S_1$ and $S_2$ shown in Figure 3.29(b) that are derived from the segmentation of the image shown in Figure 3.29(a). In order to find the tangents $T_1$ and $T_2$ to the two segments at endpoints $p_{11}(x, y)$ and $p_{21}(x, y)$, we need to consider a finite length of each segment at the respective endpoints and use this smaller segment to determine the tangent at each end point. In this process, the
number of data points considered for calculating the tangent can affect the direction of the tangent. Thus, selecting too short a segment can lead to a bias that does not take into consideration the overall curvature of the object. At the same time, selecting the entire segment can also produce misleading results.

In order to overcome this problem, we propose the following procedure: Consider segment $S_1$ with endpoints $p_{11}(x, y)$ and $p_{12}(x, y)$ shown in Figure 3.30. First, determine three points on the original segment as follows: Given the two endpoints $p_{11}(x, y)$ and $p_{12}(x, y)$ of the segment, we fit a straight line $l_{12}$ to these points and find the perpendicular bisector $h$ of this line. Next, the intersection point $m_1(x, y)$ of $h$ and the segment $S_i$ is found by solving the equation of the line representing the bisector $h$ at all points on the segment $S_i$. Given the three points $m_1(x, y)$, $p_{11}(x, y)$ and $p_{12}(x, y)$, we can fit exactly one circle through these three points. The result of this process is shown in Figure 3.30. After circle fitting, it is possible to derive the equation of a tangent to the circle at any given point as follows: Let $r_0(x, y)$ be the center of the given circle and $p_{11}(x, y)$ be a point on the circle. Given $r_0(x, y)$ and $p_{11}(x, y)$, we can fit a line through these two points that lies on the radius of the circle. Let $m$ be the slope of this line and $c$ be the intercept. The line equation is
then given by

\[ y = m_0 x + c_0 \]  \hspace{1cm} (3.2)

Using basic trigonometry, we know that the tangent to a circle at a given point is perpendicular to the line connecting this point to the center of the circle. Thus, using Equation 3.2, we can represent the tangent as

\[ y = -\frac{1}{m_0} x + c_1 \]  \hspace{1cm} (3.3)

Equation 3.3 can be solved for \( c_1 \) given the slope \( m_0 \) and the point on the circle \( p_{11}(x, y) \). Finally, given the tangent at point \( p_{11}(x, y) \) and the line segment joining the endpoints \( p_{11}(x, y) \) and \( p_{21}(x, y) \) we can now calculate the angle between these two lines by representing them as directed vectors \( v_1 \) and \( v_2 \) using the formula:

\[ \theta = \arccos \left( \frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|} \right) \]  \hspace{1cm} (3.4)
Table 3.1: Calculation of co-circularity metric: Results

<table>
<thead>
<tr>
<th>Vectors being compared</th>
<th>Calculated angle</th>
<th>Co-circularity Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>((T_1, L_1))</td>
<td>(10.96^\circ)</td>
<td>(11.6^\circ)</td>
</tr>
<tr>
<td>((T_2, L_1))</td>
<td>(22.56^\circ)</td>
<td></td>
</tr>
<tr>
<td>((T_3, L_2))</td>
<td>(36.41^\circ)</td>
<td>(3.47^\circ)</td>
</tr>
<tr>
<td>((T_4, L_2))</td>
<td>(32.94^\circ)</td>
<td></td>
</tr>
</tbody>
</table>

\(v_1\) and \(v_2\) are the vectors representing the tangent to the circle and the line segment \(l_{12}\) respectively. Applying these procedures to the segments \(S_1\) and \(S_2\), we get the tangents as shown in Figure 3.31. The angles between the tangents and the lines connecting the corresponding endpoints of the segments are listed in Table 3.1. It is observed that the co-circularity metric has a lower value when we consider the end points \(p_{12}(x, y)\) and \(p_{22}(x, y)\) for testing co-circularity of the segments \(S_1\) and \(S_2\) as compared with \(p_{11}(x, y)\) and \(p_{21}(x, y)\). We can thus reasonably reach the conclusion that these segments are part of a larger circles and are segments of the same Bowman’s space surrounding a glomeruli.

Finally, we use one additional metric that we call the directionality of a curve. We define directionality of a curve as follows: Consider a curve \(S\) as shown in Figure 3.32. Let \(L\) be the line segment connecting the two endpoints \((x_1, y_1)\) and \((x_2, y_2)\) of the curve \(S\) and \((x_m, y_m)\) be the mid-point of line segment \(L\). First, we find the intersection point \((x_i, y_i)\) of the perpendicular bisector of the line segment \(L\) and the curve \(S\). The directionality of the curve \(S\) is then defined as the direction of motion of the vector connecting point \((x_i, y_i)\) to \((x_m, y_m)\).

The goal of using the directionality metric is to find all pairs of objects that are potentially part of the Bowman’s space surrounding the same glomeruli. Consider the
segmented regions shown in Figure 3.33. Objects labeled 2 and 3 are convex in shape and are oriented such that they face away from each other. A simple method of determining pairs of objects that should be considered for grouping is to fit a parabola to each object. The orientation of the parabola can then be used to determine whether two objects should be tested for perceptual grouping. The parabolas fit to objects 2 and 3 in this figure are shown in blue and red respectively. The parabola corresponding to object 2 is oriented such that it contains object 1 and no other objects. Similarly, the parabola fit to object 3 contains all objects except objects 1 and 2. We can thus use these parabolas to perform an initial grouping of the segmented objects into two sets: one containing objects 1 and 2 and the second containing 3, 4, 5 and 6.

Using the Gestalt properties discussed here, the perceptual grouping of objects obtained from color based segmentation proceeds as follows:

- Determine the optimal \( \chi \) shape of all objects in the segmented image
- Skeletonize objects using fast marching method and obtain a 1-pixel curve representation of objects in the image
- Using the directionality of each curve thus identified, form unique pairs of objects to be considered for grouping
• Eliminate all pairs of objects that do not satisfy the proximity criteria by checking whether objects are within $80\mu$m of each other

• Eliminate connections between end points of object pairs if the co-circularity metric is too large - In this work we have used a value of $30^\circ$ for the cutoff value

3.5 Results

The proposed technique for color based segmentation and perceptual grouping of objects was applied to test images extracted from H&E stained mouse tissue biopsies. An example of the final result is shown in Figures 3.34 - 3.36. In each of these figures, the objects outlined in blue were not grouped with any other objects. The objects outlined in red, green or orange represent those objects that were assigned to the same glomeruli using the grouping process described in Section 3.4.5.

From these results, it can be seen that the approach proposed here can successfully group together objects that represent the Bowman’s space around the same glomeruli.
Figure 3.34: Example result: Objects in green and red were grouped by the perceptual grouping technique described here.

Figure 3.35: Example result: Objects in red and green were grouped by the procedure described in Section 3.4.5.
Figure 3.36: Example result: Objects in green, blue and yellow were grouped by the perceptual grouping technique described here.

However, this approach can also fail in some instances. An example of this is shown in Figure 3.37. Consider objects 1 and 2 shown in Figure 3.37(a). Grouping of these objects does not succeed due to two main reasons:

**Thickening of objects:** This is shown in Figure 3.38(a). The objects in the image show an increase in the width towards the end of the detected segment. This has the effect of adversely affecting the skeleton of the object as shown in Figure 3.38(b). In this particular case, there is a change in the curvature of the object skeleton towards the section of the object that is significantly thicker. Some of this effect can be minimized by using the circle fitting approach described in Section 3.4.5. Examining the circle shown in red in Figure 3.39, we can see that the tangent to the circle at
point $p_{11}$ (shown in green) is oriented in a direction that is influenced by the entire curve and not just the section of the object where the thickening occurs.

**Objects curving on themselves:** This phenomenon can be observed at point $p_{12}$ in Figure 3.39. The skeleton of object 1 at point $p_{12}$ curves inwards towards itself. This has the effect of influencing the circle fit to the object such that the resulting circle is smaller in radius and consequently not a true representation of the object being examined. Consider the tangents $T_1$ and $T_2$ at points $p_{12}$ and $p_{22}$ respectively as shown in Figure 3.39. Given that $l_{12}$ is the line joining the endpoints $p_{12}$ and $p_{22}$, we calculate the angles between the tangents and this line, giving us the result that $\theta_{p_{12}} = 2.98^\circ$ and $\theta_{p_{22}} = 133.77^\circ$. The co-circularity metric in this case is $\theta_{p_{12}} - \theta_{p_{22}} = 130.79^\circ$, which is significantly larger than 0, leading to a conclusion that the two curves are not co-circular.

One possible approach to avoid these problems might be to use the convex hull of the object to determine orientation and curvature of the objects. In Figure 3.40, the blue line represents the border of the convex hull of the object shown. Since we are more interested in convex objects, the convex hull will always have one straight edge. The edge of the convex hull opposite to the straight edge will typically be curved and may provide a better representation of the curvature and orientation of the objects.

While it is possible to identify glomeruli in H& stained sections of kidney biopsies, this is a challenging problem due to the fact that the main identifying landmark for the glomeruli consists of a lack of tissue or staining around the objects of interest. Additional feature from the tissue need to be derived to improve the detection of glomeruli. One of the features that might be used are the cells or cell nuclei in the
Figure 3.37: Proposed approach fails to connect the disjoint objects shown here: Different colors represent objects that were not grouped together.

(a) Red rectangle highlights object thickening

(b) Object skeleton is affected by the thickening as shown here

Figure 3.38: Thickening of objects at endpoints can cause errors in the grouping procedure.
Table 3.11: Calculated vectors and their respective angles

<table>
<thead>
<tr>
<th>Vectors</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>((T_1, l_{12}))</td>
<td>2.98°</td>
</tr>
<tr>
<td>((T_2, l_{12}))</td>
<td>133.77°</td>
</tr>
</tbody>
</table>

Figure 3.39: Co-circularity metric calculation and it’s failure to predict co-circularity

Figure 3.40: Using convex hull to determine orientation and directionality of object

tissue. At lower magnification, the presence of dense, dark nuclei inside the glomeruli may be able to guide the detection of regions of interest.

Another approach might be to use alternative stains that are also used for kidney biopsies analysis, such as the trichrome stain and silver stains. These may be better suited for the task of glomeruli identification because of their staining characteristics. The trichrome stain is particularly effective in staining Bowman’s capsules
and can potentially provide images that are more amenable to automated glomeruli identification.
Chapter 4: MULTIPROCESSOR AND GPGPU COMPUTATIONS FOR EFFICIENT ANALYSIS OF WHOLE-SLIDE IMAGES

4.1 Introduction

Developments in microprocessor technologies have resulted in most processors having multiple computing cores in a single chip. As a result, today's distributed memory high performance computers (HPCs) have multiple CPUs (2-4) in each node, with each CPU having multiple cores (2-8). The typical programming methodology for such distributed memory HPCS is using some form of a message passing paradigm, typically MPI. On the other hand, General Purpose Graphical Processing Units (GPGPUs or GPUs) are emerging as an alternative architecture for many computationally intensive tasks, including signal processing. GPGPUs have large number of processor cores (up-to 240 in some NVIDIA® GPUs) and are typically programmed using threads. MATLAB® is the de-facto language of choice for algorithm development in signal and image processing and has been traditionally used on desktop systems. Recent years have seen a surge of interest in leveraging multi-processor and multi-core systems from MATLAB. Parallel MATLAB has been actively developed over the past several years, and there are several commercial and academic versions
available [74–79]. Using MATLAB with GPGPUs is a relatively recent development, and the products are not as well developed. The options for multi-core GPGPUs are: (a) create and compile CUDA based mex functions [80] or (b) use MATLAB add-ons such as Jacket [81] or GPUmat [82] which aim to accelerate MATLAB functions. Signal processing algorithm developers who use MATLAB need to know the different options and tradeoffs to stay productive.

In this chapter, we discuss the different multi-processor MATLAB choices: (a) Parallel Computing Toolbox™ and the MATLAB® Distributed Computing Server™ [83], (b) Star-P from Interactive Supercomputing Inc. [84]; and (c) pMATLAB/bcMPI from MIT Lincoln Laboratories/Ohio Supercomputer Center [85,86]. We then look at different multi-core MATLAB choices for (a) CUDA based mex functions (b) MATLAB add-ons. For each of these technologies, we compare individual programming effort and performance improvements observed with popular signal processing kernels and applications. The main message for the reader is that it is possible to exploit today’s multi-core and multi-processor systems to effectively simulate signal processing problems that are large in memory and/or computation requirements, while staying in the familiar MATLAB environment. The required changes to sequential MATLAB code are usually quite small, and can be performed with ease. As multi-core and multi-processor implementations have been carried out on different systems and used for different problem sizes, the results are not compared directly.


4.2 Parallel Computing using MATLAB

4.2.1 Multi-threading in MATLAB

The simplest approach to leveraging multiple processor cores in MATLAB is through the use of multi-threading. Since MATLAB supports multi-threading natively [87], this approach is a simple, non-intrusive way to leverage multiple cores on a system. This type of multi-threading can be broadly compared to the OpenMP [88,89] approach to parallelism. The built-in multi-threading in MATLAB does not require any intervention on the part of the user and is enabled by default. However, the maximum number of parallel threads cannot exceed the number of cores available on the system. The performance gain obtained by using multiple cores on a single system are also limited and vary based on the specific computation as well as the data size. Fig. 4.1 illustrates this point. On a 16 core system, a maximum speedup of slightly over 7 was seen for the multiplication and \texttt{sqrt} operations. Conversely, the trigonometric function \texttt{sin()} has a speedup of slightly under 3. This test was performed on a 4-socket Quad core AMD system with 64 GB of RAM running Red Hat Enterprise Linux.

While multi-threaded computations are the easiest entry into parallel computing with MATLAB, performance gains are usually limited. This approach should only be viewed as a first step in improving the code performance.

4.2.2 Multi-Processor MATLAB

The most common approach to overcoming the performance limitations of sequential MATLAB involves distributing an application over multiple nodes of a commodity
Figure 4.1: Relative speedup using 16 threads on a 16 core, 4 socket system with the built-in multi-threading in MATLAB

cluster. Typical performance limitations for sequential MATLAB can be broadly classified into two areas: capacity and capability. The problem of capacity manifests itself as the inability for existing hardware and software to perform the desired computations in a practical amount of time. For example, this can include parameter sweeps that may take days or weeks, limiting the range of analyses performed. Similarly, the data being collected may be so large that it is not feasible to analyze the complete data set in any reasonable manner. In these cases, while the existing hardware and software are capable of performing the desired analysis, it may not be practical to run the entire computation. The problem of capability is brought about by the actual physical limitations of the system. Thus, issues related to the total memory on a system or processor speeds may limit the amount of analysis performed. While this problem can be solved in a limited way by system upgrades, there is an upper limit to this approach that is dictated by technology and cost factors. In the case of such
problems, the problem may be split up into smaller, more manageable chunks that can be performed in parallel.

### 4.2.3 Parallel computing tools for MATLAB

There are several options for leveraging the availability of multiple processors and multiple cores to solve the performance limitations in serial MATLAB. These range from utilizing multiple cores on a single processor to leveraging hundreds of processors on a commodity cluster to split up the problem. Based on the type of analysis being done, one or more of the approaches may be ideally suited to the problem. In [74] and [78], the authors highlight various tools currently available for parallel computing in MATLAB. While these tools have been designed to minimize programming complexity, in our experience, three multi-processor MATLAB technologies stand out in terms of user base, user support, and active development: pMATLAB+bcMPI, Star-P, and the Parallel Computing Toolbox™ with the MATLAB® Distributed Computing Server™. These toolboxes enable users to parallelize algorithms in MATLAB using an embarrassingly parallel approach or through the use of distributed arrays/matrices and (with the exception of Star-P), implicit message passing between multiple MATLAB processes running on different processors.

**bcMPI**

bcMPI is an open source software library that is developed by the Ohio Supercomputer Center (OSC). bcMPI provides an alternative to MatlabMPI [90] and is geared towards large shared supercomputers. bcMPI interfaces with pMATLAB [91] from MIT Lincoln Laboratories for distributed data processing. The combination of pMATLAB and bcMPI is denoted as pMATLAB+bcMPI. pMATLAB+bcMPI uses
a layer of abstraction beyond traditional MPI calls and reduces programming complexity when compared to traditional MatlabMPI programs.

![Architecture of pMATLAB/bcMPI](image)

**Figure 4.2: Architecture of pMATLAB/bcMPI from MIT Lincoln Laboratories and Ohio Supercomputer Center**

**Parallel Computing Toolbox**

The Parallel Computing Toolbox™ along with the MATLAB Distributed Computing Server™ are commercial products offered by The MathWorks™.

The PCT provides the ability to run up to 8 MATLAB processes on a single system without the use of the MATLAB Distributed Computing Server. It thus provides a convenient environment to develop and test parallel MATLAB code locally and then scale up the same code to significantly larger number of processors on supercomputing cluster through the use of the MDCS.
Star-P

Star-P is a client-server parallel computing platform for MATLAB available from Interactive Supercomputing. Star-P supports fine grained parallel as well as embarrassingly parallel modes of operation. However, Star-P does not provide functionality for explicit message passing between the processes running in parallel. Any required inter-processor communication is performed by the software itself without any intervention from the user.

4.3 Multi-processor approaches

Parallel computing in MATLAB consists of splitting up the problem across multiple processors or multiple compute nodes in a variety of ways. This section discusses the different approaches used and illustrates each approach with an example.
4.3.1 Embarrassingly parallel approach

The embarrassingly parallel approach is one of the most commonly used methods used to split up a problem across multiple processors. This approach is ideally suited for parameter sweeps or applications in which individual tasks can be completed in an order-independent manner. For example, one may wish to analyze the effectiveness of an algorithm on a given dataset by varying the parameters of the algorithm over a wide range. In such cases, each parameter set can be farmed out to a different processor, thus reducing the total time required to complete the entire analysis. Similarly, multiple, independent data sets stored in separate files can be analyzed by splitting up the work across multiple processors.
Table 4.1: Calculating $\pi$: Sequential and parallel implementation using `parfor()`

<table>
<thead>
<tr>
<th>Sequential implementation</th>
<th>Parallel implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count = 0;</code></td>
<td><code>count = 0;</code></td>
</tr>
<tr>
<td><code>for k = 1:N</code></td>
<td><code>parfor k = 1:N</code></td>
</tr>
<tr>
<td><code>    p = rand(1, 2);</code></td>
<td><code>    p = rand(1, 2);</code></td>
</tr>
<tr>
<td><code>    if sqrt(sum(p.^2) &lt; 1)</code></td>
<td><code>    if sqrt(sum(p.^2) &lt; 1)</code></td>
</tr>
<tr>
<td><code>        count = count + 1;</code></td>
<td><code>        count = count + 1;</code></td>
</tr>
<tr>
<td><code>    end</code></td>
<td><code>    end</code></td>
</tr>
<tr>
<td><code>pival = 4 * count / N;</code></td>
<td><code>pival = 4 * count / N;</code></td>
</tr>
</tbody>
</table>

The `parfor()` command

The Parallel Computing Toolbox provides a simple way to parallelize MATLAB for-loops. The `parfor()` [83] command can be used to distribute the individual loop iterations across processors without any additional code modifications. This construct is suited for loops in which the computations are order independent. Let us consider the simple algorithm for calculating $\pi$:

- Initialize a counter to zero.
- Generate two random numbers, $x$ and $y$ in Cartesian coordinates.
- If the point $(x,y)$ lies inside the unit circle, increment counter.
- Repeat above two steps $N$ time, where $N$ is some very large number
- Calculate $\pi$ using the formula: $\pi = \frac{4 \times N}{count}$

The above algorithm can be written in MATLAB as shown in Table 4.1.

This algorithm is embarrassingly parallel and can be easily parallelized through the use of the `parfor` function provided by the Parallel Computing Toolbox. By
simply changing the \texttt{for} to \texttt{parfor}, the algorithm can be run on multiple processors, assuming that some preconditions are met. For full details on using the \texttt{parfor} construct, readers are referred to the toolbox documentation and the work in [92]. In a similar manner, suppose one needs to run the same image analysis algorithm on a large set of images, a \texttt{for-loop} can be used to process all the files and can be parallelized by using the \texttt{parfor} command.

### 4.3.2 Using distributed arrays

A second approach to parallel computing in MATLAB is through the use of distributed arrays and matrices. The concept of distributed arrays/matrices is based on the PGAS (Partitioned Global Address Space) programming model in which multiple processors share a global address space [92] and each processor can read from or write to any section of the global address space [93]. The data being processed is thus distributed across multiple processors, with parts of the data being local to each processor. This distribution of data also enables the use of large data structures which may not be practical on a single processor.

In a similar manner, the different parallel MATLAB techniques discussed here enable the user to create and manipulate arrays/matrices in MATLAB that are distributed across multiple processors or on a cluster of computers. For example, the Parallel Computing Toolbox allows the creation of distributed arrays/matrices by concatenating matrices that reside on different processors, by distributing a large matrix that initially exists on a single processor or by using custom constructors provided by the toolbox. Similarly, Star-P enables the distribution of matrices through
Table 4.2: Parallelism using Distributed Arrays: Parallel 2D Convolution using pMATLAB and bcMPI.

```matlab
dist_map = map([ Np 1 ], { }, [ 0:Np-1 ]); data = rand(4000, dist_map); locData = local(data); locData = conv2(locData, H, 'same'); data = put_local(data, locData);
```

Table 4.3: Parallelism using distributed arrays: Parallel 2D FFT using Star-P

```matlab
N = 1024;
x = rand(N, N*p);
X.fft = fft2(x);
```

the use of the \*p construct. One of the biggest advantages of this approach to parallel computing is that the data distribution is handled by the underlying library. The programmer does not need to know the details of where the data resides and can focus on the actual algorithm.

Table 4.2 shows an example of a parallel 2D convolution using pMATLAB and bcMPI. The first step in this approach is to create a map that defines the distribution of the data. In this example, the map function defined by pMATLAB is used to distribute the rows of the random matrix data across Np number of processors. Each processor then performs a convolution on the part of the data which resides in its local memory and puts the results back into the global address space. Fig. 4.5, shows
Figure 4.5: Multi-Processor vs. Sequential run time for 2D convolution: Performance of parallel 2D convolution on multi-processor system using pMATLAB and bcMPI.

Figure 4.6: Multi-Processor vs. Sequential run time for 2D FFT: Runtime curves obtained by varying problem size and parallelized using Star-P
the reduction in the total computation time for a 2D convolution kernel on a matrix of size 1024x1024.

Table 4.3 shows an example of a 2D FFT operation on a distributed matrix using Star-P. The code also illustrates the ease with which Star-P can be used to create distributed arrays using the ”*p” construct. A NxN distributed matrix is created using the MATLAB function `rand` and the FFT can be calculated by simply calling the overloaded `fft2()` function.

Fig. 4.6 shows the run-times for a parallel 2D FFT using Star-P for varying data sizes. The parallel algorithm was run on 4 systems each having a 4-core AMD Opteron processor, for a total of 16 cores. It can be seen that for small problem sizes, the parallel implementation is actually slower. This is because of the large amount of inter-processor communication that has to occur when the matrix is transposed. This also illustrates one of the pitfalls that users must be aware of when using distributed matrices. Algorithms must be designed so as to minimize re-distribution of data which can lead to a reduction in performance due to excessive communication between processors.

4.3.3 Fine grained parallelism

The third approach to parallelism involves the use of message passing similar to the traditional parallel programming paradigm. Programmers can control algorithm flow, exchange data between different instances of MATLAB running on different processors and distribute the analysis through explicit message passing between the MATLAB processes. This approach gives the programmer maximum control over the
Table 4.4: An example of fine grained parallelism using bcMPI

```matlab
my_cpu = MPI_Comm_Rank( comm );
if (my_cpu > 0)
    tag = my_cpu;
    MPI_Send(0, tag, comm, data);
else
    globalsum = 0;
    for k = 1:ncpu-1
        tag = k;
        data_k = MPI_Recv(k, tag, comm);
        globalsum = globalsum + data_k;
    end
end
```

parallel implementation of the algorithm, but can be most time consuming to develop and test.

The approach to fine grained parallelism leverages the Message Passing Interface (MPI) programming paradigm. The MPI standard [94] defines the language bindings for point-to-point message passing, collective communication, process creation and management and several other protocols required for the message passing parallel programming model [95]. bcMPI and the PCT offer MPI bindings for MATLAB. These bindings include the basic MPI functions that enable point-to-point communication between the MATLAB processes running in parallel.

Table 4.4 shows an example of point to point communication between multiple processors. In this example, all processors with rank greater than zero send their local data to the rank zero processor using the `MPI_Send` command, which is then received by the rank zero processor when it calls the `MPI_Recv` command. It should be noted here that deadlocks can occur if a processor sends data without a corresponding
MPI \texttt{Recv} from the intended recipient. This is one of the major aspects of fine grained parallelism that programmers must be careful to address.

\subsection*{4.3.4 MATLAB on General Purpose Graphics Processing Unit}

Another technique for speeding up sequential MATLAB code involves using the multiple cores of CPUs and/or General Purpose Graphics Processing Units (GPGPUs) for multi-threaded computing. The main difference between this form of parallel MATLAB and multi-processor MATLAB is that multi-core MATLAB uses threading as the underlying parallel computing mechanism. Presently, there are two examples of multi-core architectures: (a) conventional multi-core CPUs (typically with 2-8 cores) and (b) unconventional multi-core processors such as GPGPUs (with tens or hundreds of cores). For the purpose of our discussion, we will concentrate on the utilization of multiple cores of GPGPUs. This form of parallel MATLAB is relatively new and the number of options available is limited.

\subsection*{4.3.5 Graphics Processing Units}

Recent trends in hardware development have led to Graphics Processing Units (GPU) evolving into highly parallel, multi-core computing platforms. Current GPGPUs such as the Quadro FX 5600 have 128 cores and newer hardware such as the Tesla™ platform from NVIDIA® can contain up to 240 processing cores per graphics card.

CUDA™ is a parallel programming model and software environment developed by NVIDIA that enables programmers to take advantage of the multi-core GPGPU with standard programming languages [80]. CUDA provides extensions to the C
programming language that enable the programmer to write fine grained parallel algorithms that can be executed using multiple, simultaneous threads on the GPGPU. Recent work [96], [97], [98] has shown the performance gains possible through the use of CUDA to accelerate a variety of algorithms.

The CUDA programming model enables programmers to run fine grained parallel code by launching multiple threads on the GPGPU. The threads are divided into blocks that can be scheduled to run independently across the GPGPU compute cores. The ability to schedule and run multiple threads simultaneously enables code scalability with the number of cores. Complete details on the CUDA programming model can be found in [80].

As shown in Fig. 4.7, the serial code running on the CPU invokes a computational kernel which is to be run on the GPGPU. Since the CPU and GPGPU memory spaces are distinct from each other, data to be used in the computations must be transferred to the GPGPU. This can be the major penalty incurred in the process and programmers must avoid unnecessary data transfer between the CPU and the GPGPU to avoid the performance penalty. The computational kernel is executed on the GPGPU through the use of grids comprised of multiple thread blocks each of which executes on a single multiprocessor.

4.3.6 Interfacing MATLAB with GPGPUs

Several toolboxes for MATLAB have been developed to allow the off-loading of computations to the GPGPU by simply casting MATLAB data into the toolbox-defined GPGPU data type [81], [82], [99]. The availability of such toolboxes makes it very easy for researchers to try out GPGPU computing without having to write
optimized C code that can take hours to develop and debug. Scientists can focus on the research without worrying about the intricacies of the C/CUDA programming paradigm. These toolboxes, however, are currently under development and may not support every MATLAB function. The most common functions supported include 1D and 2D FFT, convolution and standard mathematical operations.

One approach to off-loading computations to the GPGPU is to use the GPGPU to perform small kernels such as FFT, convolution, FIR filtering, which are often the most time consuming operations in an application. With this approach, the researcher can remain in the familiar MATLAB environment while running massively parallel algorithms transparently on the GPGPU.

4.3.7 MATLAB plug-in for CUDA

The MATLAB plug-in for CUDA available from NVIDIA’s website [100] provides the tools necessary to convert CUDA programs to MATLAB-callable mex functions. The use of this plug-in allows programmers to write custom applications that are
optimized for the given problem. This can be a challenging and time consuming task
due to the fact that the desired code must be written (often re-written) in C.

4.3.8 MATLAB Toolboxes for GPGPU computing

Currently, three toolboxes are available for the use of CUDA from MATLAB. The toolboxes are GPGPUmat [82] and gpulib [99] which are available for free and Jacket [81], which is a commercial product. Each of these toolboxes offers the ability to off-load computations to the GPGPU by simply casting MATLAB’s built-in data types to a new data-type provided by the toolbox that represents data on the GPGPU. The simplicity of using these toolboxes should be considered carefully because the code can incur heavy penalties due to data transfers between the main CPU memory and the GPGPU memory.

4.3.9 Considerations for GPGPU computing

The use of GPGPUs for off-loading computations brings additional considerations. The typical operating procedure for GPGPU computing consists of the transfer of data from the CPU to GPGPU memory when GPGPU functions are called. This transfer of data from the CPU to the GPGPU can lead to a performance penalty. When writing CUDA programs, the programmer has significant control over the data transfers and the CUDA application should be carefully designed to minimize such transfers. In contrast, the premise of the MATLAB toolboxes available for GPGPU computing is that users can accelerate their code simply by casting the data to the GPGPU data-type and performing calculations as usual. However, this must be done carefully so as to avoid the penalty incurred during data transfers. For example, in our SSCA#3 application, the FFT kernel is called multiple times in the algorithm.
Using the GPGPU to off-load the FFT calculations is an obvious path to speeding up the application. However, it was observed that using the GPGPU did not provide the expected speedup. Upon closer examination and profiling of the code it was observed that the FFT kernel was indeed faster on the GPGPU, but the performance penalty incurred in the data transfers negated the gains.

4.3.10 The SSCA#3 application

The Third Scalable Synthetic Compact Application (SSCA #3) benchmark [101], from the DARPA HPCS Program [102], performs Synthetic Aperture Radar (SAR) processing. SAR processing creates a composite image of the ground from signals generated by a moving airborne radar platform. It is a computationally intense process, requiring image processing and extensive file IO. The proposed solution to the SSCA #3 application uses a data parallel approach to parallelization. The SSCA #3 application consists of signal processing kernels such as FFTs, Convolutions, and Interpolation. In order to parallelize the SSCA #3 application, the MATLAB profiler was run on the serial implementation. The profiler showed that approximately 67.5% of the time required for computation is spent in the image formation function of Kernel 1 (K1). Within `formImage`, the function `genSARimage` is responsible for the computationally intense task of creating the SAR image. `genSARimage` consists of two compute-intensive parts, namely, the interpolation loop and the 2D Inverse Fourier Transform. In addition, multiple 1D Fourier Transforms are computed. The interpolation loop involves iteratively interpolating sections of the SAR raw image. The number of iterations is often in the tens of thousands, and each iteration contains multiple matrix multiplications and matrix additions. The 1D and 2D FFTs
are carried out only once per genSARimage function call. The problem size (size of input image) can be increased by modifying the SCALE variable. For a SCALE value of 10, the time taken by K1 is approximately 200 seconds. Amdahl’s law [103] states that the maximum speed up of a parallel application is inversely proportional to the percentage of time spent in sequential execution. In our parallelization of SSCA #3, the function genSARimage, which accounted for 67.5% of overall execution time, was parallelized. The remaining execution time (32.5%) remains serial and, therefore, the theoretical speedup on p cores is \(1/(0.325 + (0.675/p))\). The maximum speedup possible is about 3.0.

**Multi-processor implementation**

In the multi-processor implementation, a matrix F (which is interpolated to give the output image, see Table 4.5) is distributed as contiguous blocks of columns across
Table 4.5: Code Snippets for \textit{genSARimage()}: Serial Version

\begin{verbatim}
F = single(zeros(nx, m));
spatial = ifft( ifft( F, [], 2) );
\end{verbatim}

Table 4.6: Code Snippets for \textit{genSARimage()}: Parallel Version using pMATLAB + bcMPI additions

\begin{verbatim}
pFmap = map([1 Ncpus], {}, [ 0:Ncpus-1 ]); pF = zeros(nx,m,pFmap);
pFlocal = ifft(pFlocal, [], 2);
pF = put local(pF, pFlocal);
Z = transpose grid(pF);
Zlocal = ifft(local(Z), [],1);
Z = put local(Z,Zlocal);
Z = agg(Z);
spatial = abs(Z)';
\end{verbatim}

all processors. The code within the interpolation loop remains functionally equivalent with the parallel version altered such that each processor performs its calculations on a smaller, local part of the global F matrix. After the interpolation loop, the 2D IFFT is carried out through the use of pMatlab’s \texttt{transpose.grid} operation which changes the distributed F matrix from a column to row distribution. A snippet of the required changes are shown in Table 4.6 (for variables of interest). The absolute performance times and relative speedups for image formation are given in Figure 4.8. For this application, nearly 67.5\% of the code was parallelized by increasing the number of Source Lines of Code (SLOC) by 5.5\% (approximately 50 additional lines of code).
Table 4.7: Algorithm run time before and after GPGPU porting

<table>
<thead>
<tr>
<th>Operation</th>
<th>Pre-GPU Time (sec.)</th>
<th>Post-GPU Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>45.54</td>
<td>18.18</td>
</tr>
<tr>
<td>Windowing</td>
<td>24.91</td>
<td>11.18</td>
</tr>
<tr>
<td>2D FFT</td>
<td>1.64</td>
<td>0.36</td>
</tr>
</tbody>
</table>

**GPGPU implementation**

From the application analysis, it appears that using GPGPUs for these calculations would reduce overall computational time for SSCA3. To investigate this, the GPUmat toolbox was used to port the serial genSARimage function code to GPGPU enabled MATLAB code. This porting was a simple process, and required casting variables and matrices as `GPUsingle` data types. This casting moves the data from CPU memory to GPGPU memory.

After enabling the GPGPU code, it was noticed that the overall execution time post-GPGPU porting was larger than the sequential (pre-GPGPU) runtime. The MATLAB profiler was used to investigate this unexpected behavior. It was observed that as expected, functions such as FFTs, matrix multiplications, etc. showed a large reduction in computation time when using the GPGPUs. Results obtained before and after GPGPU porting for certain functions are listed in Table 4.7.

However, additional overhead due to communication between CPU and GPGPU was also observed. This overhead caused an increase in overall runtime for the GPGPU enabled code. Examples of large overhead components are shown in Table 4.8.
Due to these additional components (which are not present in the pre-GPU code) extensive modifications would be required to efficiently use GPGPUs.

4.3.11 The SQIF Application

Superconducting Quantum Interference Devices (SQUIDs) and arrays of SQUIDs or Superconducting Quantum Interference Filters (SQIFs) have a wide variety of applications [104]. SQUIDs are the world’s most sensitive detectors of magnetic signals (sensitivity in femto-Teslas) and are used for the detection and characterization of signals small enough to be virtually immeasurable by any other known sensor technology. They have applications in the detection of buried facilities from space, and the detection of weak signals in noise limited environments. The SQIF application is intended to solve large scale problems for the study and characterization of interference patterns, flux-to-voltage transfer functions, and parameter spread robustness for large SQIF loop size configurations and SQIF array fault tolerance. The technical background for the SQIF application can be found in [104]. The particular application developed was intended to run the SQIF program in an optimized fashion to either (1) reduce run-time or (2) increase the size of the problem being solved. The SQIF application involves iteratively solving ordinary differential equations as outlined in [104]. Application of the MATLAB profiler on the SQIF application using

<table>
<thead>
<tr>
<th>Operation</th>
<th>Overhead Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deleting GPU Single</td>
<td>17.7</td>
</tr>
<tr>
<td>Assigning GPU Single</td>
<td>13.6</td>
</tr>
<tr>
<td>Subscripting GPU Single</td>
<td>10.2</td>
</tr>
</tbody>
</table>
100 SQUIDs yielded a runtime of approximately 20 minutes. A detailed analysis showed most (approximately 88%) of the time spent in the function evaluations for the differential equation. Optimization was carried out on this function. Further review of the profiler results showed a linear increase in the time taken by the code as the number of SQUIDs was increased. Results of the multi-processor approach were obtained using pMATLAB+bcMPI on the Ohio Supercomputer Center’s AMD Opteron “Glenn” cluster.

The SQIF application was also parallelized using CUDA and results are discussed in the following sections. Since the GPGPU computations are performed in single precision while the MATLAB computations are in double precision, direct comparisons between the two is not strictly valid. However, the performance numbers of each technology help illustrate the gains possible. The multi-core GPGPU implementation requires a significant amount of programming effort for the large gains observed and this is a tradeoff that must be evaluated before choosing the technology to be used.

Multi-processor Implementation

At its core, the SQIF application involves solving a partial differential equation over a desired time range. At each time step, flux is calculated for each SQUID element in a vector based on its adjacent neighbors. Thus, in a fine-grained parallel implementation, each processor needs only one data point from its left and right neighbors. Figure 4.9 shows the idea behind the fine grained parallel implementation in MATLAB. As shown in the figure, the input data of length n is distributed across P processors with an overlap of 1 data point with each neighbor. The first and last elements of the output data are special cases and are calculated separately.
At the beginning of the evaluation of the differential equations, input data is distributed across P processors with the required overlap. At each step of the differential equation evaluation, a small amount of data exchange occurs between processors. For example, as shown in Fig. 4.9, Processor 2 receives data from 1 and sends data to Processor 3. In the pMATLAB implementation, this communication must occur across different compute nodes of the cluster over the Infiniband network. While the Infiniband network offers bandwidths up to 10 GB/s, the communication overhead increases significantly as the number of processors is increased. The key to achieving a good speedup is to ensure that the computation to communication ratio is large.

The time required for the SQIF application increases non-linearly as the number of devices being simulated increases as shown in Figure 4.10.

**Multi-Core Implementation using CUDA**

As described in the previous section, the SQIF application can see the most performance gains from a fine-grained parallel implementation of the algorithm. In this approach, each processor performs most of its calculations independently, but also needs to exchange data with at least one processor. The CUDA implementation of the SQIF application involved a translation of the MATLAB code into CUDA-enabled C code. In this implementation, many hundreds of threads are launched on
the GPGPU for performing the calculations. For a simple comparison, each thread can be considered as a single MATLAB process used in the pMATLAB implementation. The main difference here is that each thread only calculates a single data point in the output. Since the entire algorithm data also resides on the GPGPU, the overhead necessary to access the memory is significantly lower than the overhead in communicating over the network interface in the pMATLAB implementation. The CUDA implementation in this case was a simple implementation without any application specific optimizations. Even with this naïve implementation, a speedup of almost 28 was observed when simulating 3600 devices. Figure 4.11 shows the runtime and speedup achieved through the CUDA based parallelization. The same plot also shows the runtime for the CPU based implementation on a single 4-core AMD
Figure 4.11: Multi-Core run-time for SQIF Application: Runtime and speedup curves for varying NSquids and parallelized using the CUDA Mex interface compared with CPU based parallelization results.

Opteron based system. The speedup observed is based on the run-time for the CPU based serial implementation.

4.3.12 Discussion

In the previous section, several signal processing kernels and their performance when ported to run on commodity General Purpose Graphics Processing Units (GPGPUs or GPUs) have been described. Sections 4.3.10 and 4.3.11 describe two applications that can take advantage of parallelism attained by using multiple CPU cores as well as the GPGPU. We also examined the use of currently available toolboxes for running MATLAB code on the GPGPU without the need for re-programming algorithms in CUDA. While these toolboxes show great potential for speeding up MATLAB code, programmers need to be careful in setting up the problem since data
transfers to and from the GPGPU can negate performance gains. Similarly, it was also seen that porting computationally expensive MATLAB code to CUDA can result in significant performance gains. The drawback to this is the time required to re-code and debug the application in C. Researchers now have access to a variety of exotic architectures that can be leveraged for significant performance gains. No single approach may be optimal for all types of applications and the choice of the appropriate approach to parallelization should be made based on the user requirements.

4.4 Analysis of whole slide tissue images

Our goal in developing a computer aided diagnosis system is to analyze the whole tissue included in a biopsy. For example, in the case of follicular lymphoma, the diagnosis is based on the number of centroblasts in ten representative follicles in the tissue. However, by analyzing all the follicles in the tissue and consequently, all tissue samples associated with a single patient, it may be possible to provide a more accurate count of the centroblasts, thus leading to a more comprehensive view of the disease progression.

There is a wide range of research currently underway in the analysis of histopathological images. Typically, algorithms are developed on small fields of view (FOV) extracted from the tissue [26] and in certain cases, applications to whole slides images are not discussed. For example, [28] used 200 x 200 pixel sections extracted from colon cancer images scanned at 40x and 100x resolution, but the problem of analyzing the entire image is not addressed. We have found that there are several challenges that have to be addressed in order to analyze whole slide images using approaches
developed on smaller images extracted from the whole slide. Among currently published research involving whole slide processing, [105] describes the development of a computer aided diagnosis system for classifying stroma development and grading of neuroblastoma that was developed using 1024 x 1024 tiles and then applied to whole slide images with an overall stroma detection accuracy of 80% and an overall classification accuracy of 90%. In [106], an adaptive, multi-resolution approach was used for neuroblastoma differentiation. Another active research area is the analysis of whole slide images for the detection of tissue folds [107, 108]. The detection of tissue folds and other artifacts such as image stitching errors [109] can be used as a pre-processing step in order to avoid regions of the image which may be damaged and could lead to erroneous results. A graph based multi-resolution approach was used in [32] to detect regions of mitosis in whole slide images of breast cancer. Another approach to whole slide image analysis involves the use of lower resolution images to generate the regions of interest which are then analyzed at higher resolutions for the final diagnosis, grading and classification procedure.

4.4.1 Challenges

The most commonly used digital pathology scanners are capable of producing images that are scanned at 40x microscope resolution. The slides used in our study were scanned with the Aperio™ScanScope XT digitizer at 40x microscope resolution. At 40x resolution, one of the images used in this study was 96,899 x 174,600 pixels in size. Since lower magnifications are sufficient for follicle detection, this image was scaled down to 4x resolution resulting in an image with dimensions of 9,690 x 17,460 pixels.
<table>
<thead>
<tr>
<th>Data</th>
<th>Memory Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB Image</td>
<td>504 MB</td>
</tr>
<tr>
<td>HSV Image</td>
<td>4 GB</td>
</tr>
<tr>
<td>Texture Feature</td>
<td>1.3 GB</td>
</tr>
<tr>
<td>Image Smoothing</td>
<td>1.3 GB</td>
</tr>
<tr>
<td><strong>Total Memory</strong></td>
<td><strong>7.1 GB</strong></td>
</tr>
</tbody>
</table>

Table 4.9: Memory required for analysis of 4x resolution CD10 image

Even with a 10x reduction in the image size, analysis of the whole slide reveals two major challenges that must be addressed:

- Physical limits of the machine - Due to the size of the images involved, a large amount of memory is required to process the whole slide image as a single matrix. Even at 4x resolution, the images can be too large to be processed as a single array. As shown in Table 4.9, simply reading in the image and calculating the color and texture features used in the analysis imposes a requirement of approximately 7.1 GB of RAM. Further analysis on the algorithm revealed that the memory usage rose to almost 27 GB during the clustering procedure.

- Computation time - Assuming that the computer has a sufficient amount of memory, the total computation time can still be excessive, making the automated analysis impractical. Analysis of the 4x image of size 9,690 x 17,460 pixels took one hour. A timing analysis of the algorithm is shown in Table 4.10.

The follicle detection described here can be viewed in terms of three broad operations - two dimensional (2D) filtering, clustering and post-processing. An analysis of the compute times showed that the filtering and clustering operations were the most time consuming steps. Table 4.10 lists the total time required for the most
time consuming operations in the algorithm as a percentage of the total algorithm run-time. In Table 4.10, the ”Block artifact removal” process involves application of the k-means algorithm to a specific region in the image, as described in Section 4.4.5.

Table 4.10: Average computation time for various operations in the serial implementation of proposed algorithm: Timings are averaged over 5 runs for all images listed in Table 4.12

<table>
<thead>
<tr>
<th>Operation</th>
<th>Percent of total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Filter</td>
<td>0.974 %</td>
</tr>
<tr>
<td>Texture calculation</td>
<td>56.722 %</td>
</tr>
<tr>
<td>K-means Clustering</td>
<td>16.33 %</td>
</tr>
<tr>
<td>Block artifact removal</td>
<td>17.141 %</td>
</tr>
<tr>
<td>Other operations</td>
<td>8.832 %</td>
</tr>
</tbody>
</table>

**4.4.2 Analysis of memory usage**

In order to process an image of size 9,690 x 17,460 pixels, we need approximately 507 MB (1 MB = 1 million bytes) of physical memory for the RGB image itself because the image is of the 8-bit unsigned integer type. Converting this image to the HSV colorspace for extracting the color features requires eight times this amount (4 GB) because MATLAB stores the HSV image in double precision format. Calculation of the texture features involves conversion of the RGB image to grayscale, requiring another 170 MB of memory. The texture feature calculated from the grayscale image is stored as a double data type and consumes approximately 1.3 GB of memory. The second texture feature consists of the output obtained by median filtering the grayscale image and when converted to the double data type, requires an additional 1.3 GB of memory. Thus, simply reading in the entire image and calculating the
Table 4.11: Memory usage for serial implementation when analyzing whole slide image of size 9,690 x 17,460 pixels

<table>
<thead>
<tr>
<th>Data</th>
<th>Memory Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB Image</td>
<td>507 MB</td>
</tr>
<tr>
<td>HSV Image</td>
<td>4 GB</td>
</tr>
<tr>
<td>Texture energy feature</td>
<td>1.3 GB</td>
</tr>
<tr>
<td>Median filter output</td>
<td>1.3 GB</td>
</tr>
</tbody>
</table>

features used for clustering requires approximately 7.14 GB of memory. A summary of this memory usage is presented in Table 4.11. The next step in the processing involves running the K-means algorithm on a feature vector constructed from the color and texture features. Profiling this computation kernel showed that the memory usage increases to over 27 GB. The maximum memory usage during the K-means computations was found to be 36.4 GB. Since these numbers are tightly coupled to the image size, any increase in the tissue size and/or the use of higher resolution images (say 8x or 16x) imposes even greater requirements on the system. Thus, processing the entire image as a single array is impractical and alternative methodologies are needed.

4.4.3 Processing whole slide image

Our goal in the implementation of a computer-aided system for follicular lymphoma diagnosis is to analyze the entire tissue image for regions of interest. The analysis of whole slide images leads to computational challenges described in Section 4.4.1. Here, we propose to develop solutions that can be used to apply a variety of standard image processing techniques to the analysis of whole slides.
4.4.4 Overcoming memory limits

The problem of memory limits described in Section 4.4.2 can be solved by simply processing the entire image in blocks of size $B_w \times B_h$ at a time and stitching together all the individual blocks to form the final output. Here, $B_w$ is the width and $B_h$ is the height of the block in pixels. By reading only a $B_w \times B_h$ block of the image, each sub-image can be easily processed and the resulting blocks stored in a logical array that requires significantly lower amount of memory. Thus, for an image of size $w \times h$ pixels, the resulting binary image will only need $w \times h \times 8$ bytes of memory. Using the earlier image size of 9,690 x 17,460, the resulting binary image of the same size will only require 170 MB of data when it is stored as a single logical array.

While block processing of the whole slide image solves the memory limitations on a single machine, it also poses additional challenges. The first challenge is the choice of the block size used for processing the image. Block sizes 256x256, 512x512, 1024x1024 and 2048x2048 were tested. We chose 512 x 512 as the block size because using smaller block sizes results in follicles being artificially split across blocks while larger block sizes can cause merging of regions which can be difficult to detect and fix. The use of small blocks also leads to a larger incidence of grid artifacts which require further post-processing, thus adding to the total computation time. Additionally, the use of larger block sizes also imposes a greater memory requirement on the system.

The second challenge is the introduction of block artifacts in the results. The block artifact arises because not all follicles fit completely inside each block of the image being processed. This is due to large variations in follicle sizes and the fact that large follicles may be split across successive blocks in the image. The block artifact manifests itself in the form of objects that have artificially straight boundaries.
Figure 4.12: Detection of grid artifact in whole slide image: Artificially straight object borders indicate the presence of grid artifacts produced as a result of block processing of large image.

Figure 4.12 illustrates this problem. The green lines in Figure 4.12(a) represent the boundaries of the 512 x 512 sized blocks used to process the image. It can be seen that the grid lines pass through the objects labeled 2, 3 and 8 without introducing artificial edges in the object boundaries. These objects do not need further post-processing. However, objects 4, 6 and 7 display straight edges that are an artifact of the block processing and need to be re-processed.

4.4.5 Identifying and removing block artifact

In order to remove the block artifacts, we propose a procedure that uses morphological reconstruction [110] to first identify those objects that touch the tile borders. This is achieved by first generating a binary image with a grid whose spacing equals
the block size used for processing the image. The green lines in Figure 4.12(a) represent the grid overlaid on the image being processed. Using morphological operations, we isolate only those grid borders that touch the objects in the image. In many cases, grid lines pass through objects as seen on object 3 in Figure 4.12(a). In case of object 3, the grid border does not affect the overall object border itself. However, in contrast, object 7 in the same figure is severely affected. The grid borders touching object boundaries are identified as shown in Figure 4.12(b) after the appropriate application of morphological erosion, dilation and logical exclusive-or operation. Finally, a morphological reconstruction of the original binary image shown in Figure 4.12(a) using the binary image shown in Figure 4.12(c) as the marker results in an image that only contains the objects that require further processing. These objects are shown in Figure 4.12(d).

Thus, referring to Figure 4.12(a), objects 2, 3 and 8 are eliminated since the grid line pass through these objects. This leaves us with the grid lines that touch object boundaries and using this image as a marker, the morphological reconstruction of the mask image gives us the objects that need to be post-processed. Each object is then re-processed to eliminate the artificial border created by the block processing step. Since the feature vectors used for clustering have already been computed, the re-processing step involves running the K-means algorithm on the region of interest. It is important to note that the re-processing is done on the object by including extra border regions in an attempt to ensure that the complete object is identified. This extra processing can add a significant amount of time to the overall algorithm run-time.
4.4.6 Overcoming time limitations

Because of the excessively long time required to process whole slide images, we have implemented a parallel version of our algorithm. The goals of implementing a parallel version are as follows:

1. Enable the processing of large whole slide images by reducing the memory footprint of the algorithm - By utilizing multiple systems, the total memory required for the analysis can be distributed across several computers. Thus, each individual computer does not need physical memory in the 30-40GB range.

2. Reduce total computation time of the analysis - By utilizing multiple processors (either single machine or multiple machines), the time required for analysis can be reduced significantly.

Our parallel algorithms were also implemented in MATLAB©. There are several options available for parallel computing using MATLAB [74,78,111] and the popularity of MATLAB in the scientific and engineering community has led to several open source as well as proprietary solutions. Currently, there are three popular parallel MATLAB implementations that are actively developed and supported.

- **Parallel Computing Toolbox**: The Parallel Computing Toolbox™ (PCT) is a commercially supported product developed by The MathWorks™. The PCT provides the ability to use the familiar MATLAB desktop environment to develop and debug parallel code on a user’s desktop and to scale the algorithm on a cluster using the MATLAB Distributed Computing Server™.
\begin{itemize}
  \item \textbf{pMATLAB:} The pMATLAB [91] toolbox, developed at MIT Lincoln Labs, provides an open source solution to parallel computing using MATLAB. It provides the ability to create distributed matrices in MATLAB through Partitioned Global Array (PGAS) [91, 93] semantics. The underlying mechanism for communication is the MatlabMPI [90] library.
  
  \item \textbf{bcMPI:} The bcMPI [77] library is an open source toolbox developed at the Ohio Supercomputer Center. It is intended as an alternative to MatlabMPI and is mainly targeted towards large, shared supercomputers. The bcMPI library uses OpenMPI to provide MPI [95] style message passing over the high speed network on a cluster, thus giving the user control over fine grained communication.
\end{itemize}

The parallel MATLAB technologies listed here have their advantages and disadvantages. We have used MATLAB and the Parallel Computing Toolbox™ (PCT) for implementing parallel versions of our algorithm because of the ease of development and testing. The PCT provides the ability to create matrices that are distributed across multiple computers, enabling one to work on significantly larger data sizes as compared to a single machine. It also provides tools for message passing between multiple processes. [112].

4.5 Parallel implementation of algorithm

A performance analysis of our serial algorithm implementation revealed three major bottlenecks in the implementation: (1) median filtering, (2) calculation of texture measure using co-occurrence matrix and (3) K-means clustering. Out of these, the median filter and texture calculation can be implemented as 2D parallel filters. This implementation has been generalized so that any 2D filtering operation
Figure 4.13: Parallel implementation: Image is distributed across $N_p$ processors and processed in $B_w \times B_h$ blocks on each processor.
Figure 4.14: Inter-processor communication: Data exchange between processors is indicated by the blue lines. Processors need to exchange borders columns/rows of data that works on fixed sized windows can be run in a parallel fashion using our parallel filter code as described in Section 4.5.1. The K-means clustering was parallelized by simply using serial implementations of K-means on distributed matrices. Figure 4.13 shows the flowchart of the parallel implementation of our algorithm.

In order to parallelize our algorithm, our approach is to distribute the image data across multiple processors. Each processor reads in only a subsection of the image and works on the section of the image that is local to the specific processor. A small amount of communication between the processors is necessary in order to exchange padding columns/rows as described in the next section.

### 4.5.1 Parallel 2D Filtering

The median filtering and texture calculations are operations that are performed on a 2D matrix using kernels of size \( m \times m \). Typically, square windows of odd sizes are used for calculating the filter outputs. For example, in our algorithm the median
filter is applied to 45 x 45 windows and the texture calculations are performed on 15 x 15 windows for 4x resolution images. The rest of this section describes the implementation of a parallel filter that operates on $m \times m$ kernels. While the median filter is used as an example, the same approach is valid for any filter that operates on similar kernels.

A $m \times m$ median filter centered at pixel $p(i,j)$ replaces the value of $p(i,j)$ with the median value of all pixels in the $m \times m$ neighborhood around $p(i,j)$. This operation can be parallelized by using the following approach:

1. Distribute 2D image matrix across $N_p$ processors along the columns. Each processor now has $h \times w_p$ section of the image, where $h$ is the number of rows in the original image and $w_p$ is the number of columns on processor $p$.

2. All processors exchange $(m-1)/2$ padding columns of data with their neighbors. This is illustrated in Figure 4.14 where processor 2 exchanges two columns of data each with processor 1 and 3. This results in each processor having additional data to process. The red dashed lines in the figure encompass the total amount of data to be filtered by each processor.

3. Apply filter to local data on each processor. The 2D filter is applied to the padded matrix on each processor. In Figure 4.14, the red dashed lines indicate the data on each processor. Thus, each processor now applies the 2D filter to the padded array as shown here.

4. Discard padding columns and combine partial results from each processor to get the final result.
While this example uses a column-based distribution of data across processors, we have also implemented a row-based data distribution method.

Since each processor needs to exchange data with other processors, there is some communication overhead which depends on the number of processors and the window size used for the 2D filter. This communication results in a less than linear speedup as the number of processors are increased. Using this approach any 2D filter that operates on small kernels can be parallelized. Calculation of the texture energy from the co-occurrence matrix was also implemented in a parallel way using this approach.

4.5.2 K-means On Distributed Matrices

The K-means clustering algorithm is a well studied approach to data clustering [113–115]. Several parallel implementations of the algorithms have been developed [116–118] including implementations that run on graphics processing units (GPUs) [119, 120]. We have used the k-means++ [121] algorithm written in MATLAB for selecting initial centers. As implemented in our approach, the K-means clustering was parallelized by simply using serial implementations of K-means on distributed matrices. This approach is much simpler to implement and takes the advantage of much larger memory available by distributing the data across multiple processors. The parallel implementations of the median filter and the texture calculations produce the feature vectors that are used for clustering. The outputs of the median filter and texture calculations are also distributed matrices. Since each processor has a subset of all the feature vectors, the K-means algorithm can be run locally by each processor. The K-means algorithm is also run using the block processing methodology described in 4.4.4, with the block size being 512 x 512 pixels.
Table 4.12: Image sizes for data used in this study

<table>
<thead>
<tr>
<th>Image</th>
<th>Size (in pixels)</th>
<th>Image</th>
<th>Size (in pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>9689 x 17459</td>
<td>Image 7</td>
<td>8859 x 9797</td>
</tr>
<tr>
<td>Image 2</td>
<td>7285 x 14565</td>
<td>Image 8</td>
<td>7977 x 9234</td>
</tr>
<tr>
<td>Image 3</td>
<td>7186 x 14565</td>
<td>Image 9</td>
<td>6910 x 9125</td>
</tr>
<tr>
<td>Image 4</td>
<td>7896 x 12185</td>
<td>Image 10</td>
<td>7809 x 7491</td>
</tr>
<tr>
<td>Image 5</td>
<td>8657 x 10567</td>
<td>Image 11</td>
<td>7064 x 8259</td>
</tr>
<tr>
<td>Image 6</td>
<td>8425 x 10662</td>
<td>Image 12</td>
<td>7168 x 7396</td>
</tr>
</tbody>
</table>

4.5.3 Results

This study focused on the analysis of CD10 stained images of follicular lymphoma. As mentioned previously, the images used in this study were scanned using an Aperio™ ImageScope digitizer at 40x microscope resolution. All images were originally in the SVS image format developed by Aperio and is a JPEG-2000 compressed TIFF format. Images were downsampled to 4x resolution and converted to uncompressed TIFF format using the ImageScope [122] viewer provided by Aperio. Details of the downsampling technique used by the viewer is not available at this time and to our knowledge it is not possible to change the method used. A list of the 12 images used in this study and the image dimensions at 4x resolution are shown in Table 4.12. Figure 4.15 shows two of the whole slide images that were used in this analysis. The green outlines in each image shown outline the final borders of the follicles determined by the proposed algorithm.

The serial and parallel implementations of our algorithm were applied to whole slide images of follicular lymphoma. The analysis was done on the high performance computing resources (HPC) at the Ohio Supercomputer Center [123]. The cluster is an AMD Opteron based system consisting of 877 dual socket, dual core 2.6 GHz
Figure 4.15: Whole slide image analysis: Two tissues used in this study are shown here. Green outlines indicate follicle boundaries obtained from the proposed algorithm.

Opteron processors with 8 GB of RAM and 650 dual socket, quad core AMD Opteron with 24 GB of RAM. It also includes ten systems with quad socket, quad core AMD Opteron with 64 GB of RAM.

The serial version was tested on a system with a quad socket quad-core AMD Opteron processors with 64GB of physical RAM. On this system, there is sufficient memory to actually perform the analysis on the entire image by treating it as a single array. In this case, since the entire image can be processed as single array, the problem of block artifact removal does not arise. The serial algorithm was also run on a system with dual socket quad-core AMD Opteron processors with 24GB of RAM using the block processing approach described in Section 4.4.4. Finally, the parallelized version of our algorithm was run on the dual socket, quad core AMD Opteron systems.
Table 4.13: Comparison of total algorithm time in seconds for serial algorithm with and without block processing and parallel algorithm running on 12 processors and using block processing

<table>
<thead>
<tr>
<th>Image</th>
<th>Serial ($T_s$)</th>
<th>Serial with block processing</th>
<th>Parallel using 12 processors ($T_p$)</th>
<th>Speedup $T_s/T_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1220.52 ± 127.36</td>
<td>1570.13 ± 8.96</td>
<td>151.26 ± 4.87</td>
<td>8.06</td>
</tr>
<tr>
<td>2</td>
<td>855.62 ± 40.08</td>
<td>974.78 ± 7.18</td>
<td>111.52 ± 13.47</td>
<td>7.67</td>
</tr>
<tr>
<td>3</td>
<td>802.63 ± 43.86</td>
<td>953.4 ± 11.95</td>
<td>98.25 ± 6.24</td>
<td>8.16</td>
</tr>
<tr>
<td>4</td>
<td>717.53 ± 67.32</td>
<td>868.83 ± 6.76</td>
<td>86.76 ± 5.55</td>
<td>8.27</td>
</tr>
<tr>
<td>5</td>
<td>699.56 ± 98.96</td>
<td>805.26 ± 5.76</td>
<td>84.70 ± 7.19</td>
<td>8.25</td>
</tr>
<tr>
<td>6</td>
<td>713.98 ± 35.19</td>
<td>890.61 ± 7.72</td>
<td>86.99 ± 6.68</td>
<td>8.20</td>
</tr>
<tr>
<td>7</td>
<td>776.53 ± 44.5</td>
<td>806.66 ± 9.52</td>
<td>81.42 ± 5.50</td>
<td>9.53</td>
</tr>
<tr>
<td>8</td>
<td>579.31 ± 59.53</td>
<td>701.41 ± 4.74</td>
<td>71.24 ± 5.34</td>
<td>8.13</td>
</tr>
<tr>
<td>9</td>
<td>543.22 ± 50.06</td>
<td>634.65 ± 10.74</td>
<td>77.03 ± 14.02</td>
<td>7.05</td>
</tr>
<tr>
<td>10</td>
<td>518.32 ± 31.65</td>
<td>559.21 ± 2.47</td>
<td>75.98 ± 23.391</td>
<td>6.82</td>
</tr>
<tr>
<td>11</td>
<td>466.91 ± 25.91</td>
<td>553.26 ± 9.43</td>
<td>64.08 ± 8.64</td>
<td>7.2</td>
</tr>
<tr>
<td>12</td>
<td>433.23 ± 20.89</td>
<td>487.51 ± 1.43</td>
<td>53.63 ± 3.34</td>
<td>8.07</td>
</tr>
</tbody>
</table>

described above. Each image was processed using the parallel algorithm with the number of processors used ranging from 2 to 12. Each image was analyzed using column distribution across all processors. The parallel algorithm was run 10 times per image, per set of processors.

Table 4.13 shows a comparison on the total algorithm time for the original serial version, the serial version with block processing and the parallel version when run on 12 processors. In this table we observe that the addition of block processing results in an increase in the total processing time as compared with the serial version without using block processing. This is because of the fact that the block processing approach leads to the introduction of grid artifacts that have to be removed as described in Section 4.4.5. This removal process is carried out with additional image data included on all sides on the object being re-processed - thus leading to an increase in the total
compute time for the serial algorithm. In the images used in this study, the use of block processing in the serial implementation of the algorithm led to a maximum of 29% increase in the total compute time for Image 1 in Tables 4.12 and 4.13. However, the use of block processing enables the processing of larger images that will require significantly greater amounts of memory for processing.

4.5.4 Parallel algorithm performance

From Table 4.13 we also observe that a non-linear speedup is observed when using 12 processors. There are primarily two causes of this non-linearity. The first is the introduction of the grid artifacts which results in additional processing. These extra computations are dependent on the amount of local data being re-processed and do not scale linearly with the number of processors used. The second cause of the non-linear speedup is the imbalance in the data distribution across processors.

<table>
<thead>
<tr>
<th>Processor Number</th>
<th>Non-background pixels (in millions)</th>
<th>Number of calculations for 4 centers (in millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.4467</td>
<td>45.8670</td>
</tr>
<tr>
<td>2</td>
<td>15.2282</td>
<td>60.9128</td>
</tr>
<tr>
<td>3</td>
<td>14.9236</td>
<td>59.6944</td>
</tr>
<tr>
<td>4</td>
<td>8.5181</td>
<td>34.0725</td>
</tr>
<tr>
<td>5</td>
<td>8.0459</td>
<td>32.1836</td>
</tr>
<tr>
<td>6</td>
<td>10.8125</td>
<td>43.2502</td>
</tr>
<tr>
<td>7</td>
<td>4.9670</td>
<td>19.8682</td>
</tr>
<tr>
<td>8</td>
<td>4.1810</td>
<td>16.7242</td>
</tr>
<tr>
<td>9</td>
<td>11.4379</td>
<td>45.7516</td>
</tr>
<tr>
<td>10</td>
<td>10.3697</td>
<td>41.4788</td>
</tr>
</tbody>
</table>
The manner in which each image is distributed across the processors leads to a different number of non-background pixels on each specific processor. For example, consider the case of 10 processors using the image of size 9689 x 17459 (Image 1 in Table 4.12) and the image being distributed along the columns. Each processor uses only non-background data for k-means clustering. In this particular case, the number of non-background pixels on each processor and the corresponding number of calculations per k-means iteration are as shown in Table 4.14.

From this table, we see that processor 8 has to perform approximately 16.7 million calculations per iteration of the k-means algorithm, whereas processors 2 and 3 have to perform almost 4 times as many calculations per iteration. Thus, the time required for processing the whole slide image could potentially be dominated by the time required for the k-means step on processors 2 and 3, in addition to the actual number of iterations required for convergence. The solution to this problem is to distribute the image across all the processors in a block cyclic manner so that the regions containing the background of the image are not concentrated on a single processor.

Figure 4.16 shows the run-time observed for the parallel texture energy calculations and the parallel median filter with increasing number of processors for two of the largest images used in this study. It can be seen that increasing the number of processors has a more significant effect on the texture energy calculations. Starting at 10 processors, the computation time starts to level off. This is caused by the fact that each processor now has a much smaller amount of data and the added benefit of an increased number of processors does not translate into a significant speed gain.
Figure 4.16: Run-time for parallel median filter and texture energy calculation for two images used in the study

Figure 4.17: Run-time for parallel k-means calculations for two images used in the study
Figure 4.17 shows the time required for the parallel k-means implementation for two of the largest images used in this study. The result of the K-means algorithm is used to threshold each image block which identifies follicles in the image.

Figure 4.18 shows the average time required for the parallel implementations of the median filter, texture energy calculations and k-means clustering for four of the largest images used in this study. It can be observed that the time required decreases with the number of processors used for the parallel algorithm. The average time for the entire algorithm is shown in Figure 4.19. The average time was calculated across all images and all processor combinations. The timing clearly shows a trend of
Figure 4.19: Computation time for parallel algorithm: Average time for entire algorithm calculated over 10 runs of the algorithm across all images used in the study decreasing time required to process whole slide images as the number of processors is increased.

4.6 Conclusion

In this chapter we have presented an efficient parallel implementation of an automated algorithm for detecting follicles in IHC whole slide images. The algorithm was developed and originally tested on small images. We have presented the challenges encountered in applying the same algorithm to find follicles in images of whole slides and we have presented the use of parallel computing as one potential solution. The use of multiple systems for processing one image can help reduce the total compute time required for the analysis as well as overcome physical memory limits on a single system. The ability to use high level languages such as MATLAB to quickly prototype algorithms and test on whole slides using parallel computing can be a powerful
tools in automated computer-aided diagnosis. While the current implementation of
the parallel algorithm has been found to greatly reduce the computation time, the
algorithm does not perform load balancing across multiple processors. Investigation
into this aspect of the parallel implementation is needed to improve performance.
This research aims to develop computer aided tools for analysis of high resolution images of tissue biopsies. In order for such a system to be used in research or clinical practice it must be easy to use, intuitive and non-intrusive. Users must also be able to interact with the system without the need for any specialized training. Finally, with the large data sizes involved, it is prohibitive to have the data on every computer or device that the end user wishes to use. Thus, a web-based system that takes advantage of the ubiquity of the internet and the proliferation of media consumption devices such as tablets and smartphones presents a possible solution to enabling wide acceptance of computer aided systems in medical research as well as education.

As a final contribution of this research, we propose building such a prototype system that will allow researchers to use a simple web based interface for viewing high resolution images via a browser as well as leveraging high performance computing resources for analyzing large amounts of data. Providing a web based interface can make it possible for anyone to run different analyses and compare results easily, while also helping improve the system by pointing out mistakes made by the automated analysis.
Such a web based system for analyzing and viewing high resolution pathology images has potential applications in teaching as well as research. For example, an instructor may use a web based system to share tissue samples with hundreds of students at an educational institution. This will allow students to view and study the images at their convenience and as often as they wish without the need to carry glass slides or reserve time in labs equipped with the requisite microscopes. Faculty can also use this system to assign tests to students and through usability studies, assess the areas where students make common mistakes. Additionally, students may also share questions with each other and with the instructor.

5.0.1 Serving High Resolution Images over the Web

High resolution images obtained from scanners such as the Aperio SlideScan system can be very large, with dimensions of the order of 100,000 x 100,000 pixels. Currently, the preferred method of visualizing these images is through use of vendor provided image viewers. For example, the images used in this research were scanned using Aperio scanners and can be viewed using the ImageScope software distributed by Aperio Inc. However, this software has certain limitations that make it difficult to use in a collaborative, research environment. Namely,

- ImageScope can only be used on the Windows platform: Use on other operating systems such as Linux or Mac OS X is supported through the use of Windows Virtual Machines which makes it impractical for some users.

- Image data has to reside on the same system as the ImageScope software itself: This makes it impossible to view the data on handheld devices such as smartphones or tablets.
• Addition of image processing routines, while possible, is also impractical due to the Windows-only nature of the software. Licensed image analysis plug-ins are available from the vendor, but the costs can be high, making it an unattractive proposition.

• Additional web server software (WebScope) must be purchased in order to view images that are stored on remote servers. This added expense may not be practical for all facilities. It is also not clear whether the web server software (WebScope) allows the integration of image processing algorithms developed by users.

Thus, there exists a need for a simple, intuitive way to view high resolution histopathological images on a variety of platforms and a wide range of form-factor devices without the need to store the images themselves on the same device. Additionally, for imaging researchers as well as clinicians, there needs to be a simple way to enable users to run automated image processing algorithms on the tissue images along with the ability to view results, annotate images and easily share results with their collaborators.

Virtual microscopy can be described as the use of computers for viewing, navigating, manipulating and annotating histopathology slides [124]. The use of virtual microscopy has seen a growing rate of adoption in medical education [124–129] with positive outcomes and assessments from faculty as well as students. Developments in the area of geospatial imaging and the availability of services such as Google Maps and Bing Maps has made it possible to serve up very high resolution images seamlessly over the web. There are several services that use standard web technologies that facilitates the viewing of high resolution images. Notable among them are Google Maps,
Bing maps, OpenLayers [130], Seadragon [131], DeepZoom [132] and IIPImage [133]. While the exact details of each service and server implementation are different, all of them use the same basic principles for serving images as described in the following section. In this research, we follow the approach developed by Triola & Holloway [124] for the viewing and annotation of Aperio SVS [134] images over the web.

5.0.2 Architecture

The basic idea behind web mapping techniques is to represent the images being served as a map and to only display the section the image that the user is currently viewing at the given resolution. Consider the whole slide image shown in Figure 5.1(a). At 40x resolution this image is 103,767 x 84,337 pixels in size. Using JPEG2000 compression, this results in an image file of size 6.8GB and the uncompressed file size is 24.45 GB. Such data sizes make it impractical to easily share the images with a wide community. However, using web technologies, one can serve up only the section of the image that the user is currently viewing, at the given resolution. This is shown in the browser screenshot in Figure 5.1(b) where the region highlighted in green in Figure 5.1(a) is displayed in a browser at 20x resolution.

From an image processing perspective, this is akin to creating an image pyramid that contains a multi-resolution decomposition of the image. When a user views an image, only the section of the image in the user’s field of view is extracted from the pyramid at the given resolution and rendered on the user’s device. The two image formats that support pyramidal reductions inside a single image file are TIFF and JPEG2000. The TIFF format allows the creation of a single image with multiple “pages”, with each page containing a different image. Similarly, the JPEG2000 format
supports for a multi-resolution decomposition of an image in the same file. Both of these formats have the advantage of being able to read a sub-section of the image without reading the entire image into memory.

While it may be possible to create such an image pyramid and store it inside a single image in the form of a multi-page TIFF or a JPEG2000, there are multiple system-level issues that must also be solved when attempting to serve such an image over the web. For example, in order to read a section of an image and send it to a user’s browser, it is necessary to write server-side code (CGI) that is responsible for reading the image. Due to the stateless nature of web pages and the architecture of CGI servers, this can present a significant load to the system. One successful implementation of this system is the IIPImage [133] library that has been shown to be very useful in serving high resolution images. Another factor that can affect the use of multi-page TIFF images is that the TIFF image with multiple resolutions embedded inside it can also be very large and disk utilization becomes a severe issue as more and more images added. While using JPEG2000 compression or TIFF images with JPEG2000 (such the Aperio SVS format) can potentially alleviate this problem, the cost of licensing commercial JPEG2000 libraries can be an additional hurdle that may not be acceptable in all situations. Although open source JPEG2000 libraries exist, they may not have the same stability and decoding performance as that of commercial libraries. While we have not found research papers quantifying JPEG2000 decoding performance of open source and commercial libraries, it is our intent to perform such a comparison in the future.

A more common approach to solving the issue of serving up these images to a browser is to use the following steps:
Figure 5.1: Whole slide image of size 103,767 x 84,337 pixels at 40x microscope resolution: Green rectangle highlights the region of interest being displayed in a browser at higher resolution. Image courtesy of Dr. Christopher Pierson, OSU College of Medicine
1. Determine the number of resolution levels possible for a given image: This is done using a pyramidal reduction of the image starting at the highest resolution level and scaling down until the entire image thumbnail can fit within 256x256 pixels

2. For each resolution level:

   - Split image into tiles of appropriate size (Google Maps uses 256x256 pixel images)
   - Organize images in the manner expected by the mapping library

Once an image has been split and organized into the necessary tiles, the web server can then directly server up the images that are necessary when a user loads the appropriate HTML page. Although this is the simplest method for building such a system, it also has certain disadvantages. Specifically, the number of images tiles generated from a single image can be in the thousands. For the image shown in Figure 5.1(a), the total number of files generated was 178,831 with 2.7GB disk space required for these image tiles when compressed using JPEG compression. This is clearly not sustainable as tens or hundreds of scanned tissue images are added to this system. Tiling individual images thus creates the problem of additional disk space as well an exponential growth in the total number of files that reside on disk. One way to solve the problem of a large number of small files on disk is to store the image tiles as binary data inside a database such as MySQL. This can help reduce the stress on the file system and by appropriate optimization may result in a performance that is comparable to having small files on disk.
Figure 5.2: Options for serving up histopathological images using a single image or multiple tiles

In summary, there are two main ways of providing web based viewing of high resolution histopathological images: (1) Using multi-resolution TIFF or JPEG2000 images, (2) Generating small tiles from high resolution images. When generating small tiles, there are two further options - storing the files on disk or in a database. Figure 5.2 illustrates these options. Finally, the advantages and disadvantages of using TIFF and JPEG2000 versions of images are summarized in Table 5.1.

5.0.3 Implementation Using Google Maps

While there are several libraries available for implementing such a system, we have decided to use the Google Maps API for our implementation. The primary reasons for using the Google Maps API was the large amount of documentation, user community support and the availability of third-party tools that make it possible to implement rich web based systems for interacting with maps and images. At the same time, by using the Google Maps standard for creating image tiles, it is also possible to use
<table>
<thead>
<tr>
<th>Method</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIFF</td>
<td>• Simple to implement with open source software</td>
<td>• Can produce large files</td>
</tr>
<tr>
<td></td>
<td>• Choice of compression schemes: LZW, JPEG, JPEG2000, uncompressed</td>
<td></td>
</tr>
<tr>
<td>JPEG2000</td>
<td>• Small file size</td>
<td>• Proprietary libraries needed for good performance</td>
</tr>
<tr>
<td></td>
<td>• Lossless compression</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of methods used to serve high resolution images over the internet using a single multi-resolution file

Figure 5.3: Example image: At 40x resolution this image is 58,224 x 61,879 pixels in size

OpenLayers, the open-source mapping library, as an alternative with minimal changes to the web setup.
Figure 5.4: Zoom level 1: Four sub-images of size 256x256 pixels. Actual tile size can be smaller depending on the size of the underlying image at each resolution.

For serving high resolution images over the web using Google Maps, the first step is to create custom tiles as per the API specification. At each resolution, the image is split into sub-images or tiles of dimension 256 x 256 pixels. Consider the IHC stained image shown in Figure 5.3. This image at 40x resolution is 58,224 x 61,879 pixels in dimensions. Thus, at the highest possible zoom, this image is split into 55,176 sub-images of size 256 x 256 pixels. Table 5.2 shows the number of tiles created at each zoom for this specific image. The lowest zoom level (level 0) has a single tile of size 256 x 256 that contains a thumbnail of the entire image. The next higher resolution zoom level contains an image that is doubled in size in both dimension and has four sub-images of size 256x256 as shown in Figure 5.4. Furthermore, as the image resolution increases, the number of sub-images is squared as shown in Figure 5.5. This process continues until the highest resolution image (40x in our case) is split into the appropriate number of sub-images.
5.0.4 Annotating Histopathological Images

In addition to the ability to display high resolution histopathological images, one of the goals of this research is to allow experts to annotate the same images and display the results of automated analyses on the image. Using JavaScript one has the ability to place markers and text controls on the images using the Google Maps API.

The Google Maps API includes functionality to place markers interactively or programatically on maps (or images in our case) using browser-based JavaScript. The challenge here is in translating the pixel-location of the markers (or other annotations) to the appropriate geospatial latitude and longitude co-ordinates used by the

Figure 5.5: Zoom level 2: Sixteen sub-images of size 256x256 pixels
Table 5.2: Number of 256x256 pixel images created at each resolution for one of the images used in this research: A total of 73,681 images are generated for all resolutions combined.

<table>
<thead>
<tr>
<th>Zoom Level</th>
<th>Microscope Resolution</th>
<th>Num. Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>40x</td>
<td>55,176</td>
</tr>
<tr>
<td>7</td>
<td>20x</td>
<td>13,794</td>
</tr>
<tr>
<td>6</td>
<td>10x</td>
<td>3,477</td>
</tr>
<tr>
<td>5</td>
<td>5x</td>
<td>899</td>
</tr>
<tr>
<td>4</td>
<td>2.5x</td>
<td>240</td>
</tr>
<tr>
<td>3</td>
<td>1.25x</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>0.625x</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0.3125</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0.15615x</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total number of files</strong></td>
<td></td>
<td><strong>73,681</strong></td>
</tr>
</tbody>
</table>

Google Maps service. This is especially critical when the goal is to overlay results of computerized analysis on the original image while retaining the original, unmodified version.

Figure 5.6: Conversion of pixel co-ordinates to latitude and longitude values: Values are computed based on a 256x256 pixel image tile

Google Maps is designed to display maps of the Earth and uses the World Geodetic System WGS84 [135] standard to specify latitude and longitude points on the world.
map. When viewing maps in a browser, only a subsection of the world is displayed in the browser thus, the longitude and latitude co-ordinates from the entire world map must be translated to corresponding pixel locations on the visible section on the map. These co-ordinates are termed as world co-ordinates. The Google Maps API accomplishes this by using the Mercator Projection [135]. It is assumed that a map at zoom level 0 consists of a single image of size 256x256 pixels. The world co-ordinates are then calculated relative to pixel co-ordinates at zoom level 0. This makes the world co-ordinates independent of the zoom level of the image. Finally, the actual pixel co-ordinates are calculated as follows:

\[
pixelCoordinate = worldCoordinate \times 2^{zoomLevel}
\] (5.1)

Using the formula in Equation 5.1, we can now programmatically place markers as well as polygons on the displayed histopathological images. Additionally, web users can add markers to the image from their browser and the co-ordinates of these markers can be translated to pixel locations in the original image. This has the advantage of allowing us to display results of image analysis algorithms as overlays on the image without modifying the original image and it also allows the end user to select regions of interest which can then be extracted for automated image analysis. The user now does not need to have any knowledge of the high performance computing resources being used at the server side. Figures 5.7(a) and 5.7(b) show examples of annotations and regions of interest created by an expert pathologist for use in teaching medical students.
Figure 5.7: Example of annotating histopathological images
The same principle can be used to display the results of any algorithms run on the original image. The steps involved in translating the results from pixel co-ordinates in the image domain to latitude and longitude values in the map domain are as follows:

- Determine the boundaries of the regions to be highlighted on the image. Typically, we are interested in showing borders of cells or tumor regions.

- Downsample the object boundaries. This step, while optional, is necessary when showing large borders because a larger number of points chosen on the object boundaries can make the resulting web pages load slower and reduce the overall responsiveness for the user.

- For each object translate pixel co-ordinates to world co-ordinates using Equation 5.1.

- Convert world co-ordinates to latitude and longitude values in the Mercator System used by Google Maps. Given a point \( p(x, y) \) in an image tile of size 256x256 pixels with the origin at the center of the image tile as shown in Figure 5.6, the conversion from pixel co-ordinates to map co-ordinates uses the procedure shown in Table 5.3.

Figure 5.8(a) and 5.8(b) show examples of displaying results of glomeruli detection. Here, the image regions were analyzed in MATLAB and the results of segmentation were converted to the corresponding map co-ordinates for display in a browser.

5.1 Results and Future Work

In this chapter we have discussed the need for an easy, portable and non-intrusive way of viewing and annotating high resolution histopathological images. The use
tileSize = 256

origin = ($\frac{\text{tileSize}}{2}$, $\frac{\text{tileSize}}{2}$)

pixels_per_longitude_degrees = $\frac{\text{tileSize}}{360}$

pixels_per_longitude_radians = $\frac{\text{tileSize}}{2\pi}$

longitude = $\frac{p(x,y)-\text{origin}}{\text{pixels_per_longitude_degree}}$

latitude_radians = $\frac{p(x,y)-\text{origin}}{-\text{pixels_per_longitude_degree}}$

latitude = $(2 \times \text{atan}(e^{\text{latitude_radians}}) - \frac{\pi}{2}) \times \frac{\pi}{180}$

Table 5.3: Pseudo code for converting point in a Google Maps image tile to latitude and longitude values based on the Mercator projection
Figure 5.8: Outlining results of image segmentation algorithm using Google Maps API: Paths around the glomeruli are shown in green, yellow and red
of mapping technology such the Google Maps API allows us to provide just such a service in a device agnostic manner. This web based interface also allows us to provide a simple way for medical professionals to harness high performance computing resources for applying image tools to histopathological image analysis. There are several research questions that still need to be answered. The first question arises from the decision to use multi-page TIFF image or JPEG2000 images. While the JPEG2000 format can provide lossless compression, open source tools may not be robust or efficient enough to justify the use of this format. However, when considering system level requirements for hosting hundreds or thousands of images, the image file size becomes more pertinent. Additionally, one can make the case that using a lossy JPEG compression to serve up images may be sufficient while at the same time retaining the original images for the purpose of image analysis. Another potential issue to consider is the licensing for Google Maps - While educational grants provide a way to use the Google Maps API at no cost, this may not always be the case. Thus, investigation into the use of an open source mapping library such as OpenLayers is needed, along with a comparison of performance of the two libraries. Finally, one also has to consider the possibility of developing native applications for portable devices such as smartphones or tablets. By using a native interface instead of a web page, we gain the ability of running small image processing tools on the device itself. This can give even more powerful tools to the researcher who can then prototype on small regions of interest on their handheld device and off-load whole slide processing to a cluster.
Chapter 6: CONCLUSION

The goal of this dissertation is to study the use of automated image analysis tools in the field of histopathology and examine the challenges that are presented by the abundance of data and the problem of making this technology more accessible to a wider community that includes medical practitioners and researchers as well as students. As digital pathology gains wider acceptance in research and clinical practice, the problems of analyzing ever increasing datasets needs to be addressed. Regardless of the actual disease being studied, there are common problems that must be addressed. From a computer aided analysis point of view, these problems can be distilled down to identification of salient features in tissues, morphometric quantification, counting structures in tissue and quantifying the amount of staining. Thus, in Chapters 2 and 3 we explore the use of image analysis for identifying features and structures of interest for two very different diseases, viz, Follicular Lymphoma and Lupus. In both of these cases, immunohistochemical stains are used to quantify disease progression. For follicular lymphoma we use texture and color features for the identification of follicles in the tissue. We also investigate the feasibility of using protein signatures obtained from mass spectrometry to identify follicles in the tissue, with promising results. The study of lupus involves two very different kinds of stains: Immunohistochemical (IHC) and H&E stains. We use IHC stained tissues
for quantifying the amount of infiltrate in the tissue biopsy. In parallel, we also propose the development of glomeruli identification in H&E stained tissue using color based segmentation. This is followed by a perceptual grouping process to identify the glomeruli by isolating the Bowman’s space surrounding the glomeruli in the tissue. This work is still being pursued and other stains are also being considered for use in the image analysis.

In Chapter 4 we address the computational aspects related to the processing of high resolution microscopy images. With image sizes in the 100,000 x 85,000 pixel range, there are several problems that need to be addressed. Even when images are downsampled significantly, the amount of physical memory and time required for analysis may be impractical for using these tools in routine practice. Thus, we explore the use of high performance and parallel computing to solve these problems. By splitting up the problem into smaller sections and using multiple processors, we can run these algorithms in a much more reasonable manner. The emergence of graphics processing units as powerful computing tools has also opened up a brand new avenue of research that can potentially accelerate the analysis even further.

Finally, these tools are not of any value unless they can be put into the hands of researchers who may not have any computer engineering background. Thus, in Chapter 5 we address the problem of disseminating the tools we develop to a much wider audience. By using freely available tools and web technology we can give digital tools to any researcher or student and make it available on the device of their choice. Faculty can use these tools in classroom setting for making significantly increased resources such as sample of interesting cases available to the entire student body without the need for providing an equal increase in physical laboratory resources.
such as microscopes or glass slides that may get easily damaged. Such web based access to digital pathology coupled with high performance computing resources can greatly accelerate the pace of research by allowing non-engineers to run analysis on tissue image and visualize the results without the need to become experts in image processing.
Bibliography


[19] Mikhail Teverovskiy, Vinay Kumar, Junshui Ma, Angeliki Kotsianti, David Verbel, Ali Tabesh, Ho-Yuen Pang, Yevgen Vengrenyuk, Stephen Fogarasi, and


[76] Ashok Krishnamurthy. Parallel and Distributed MATLAB applications in Signal and Image Processing.


[81] Jacket v 1.01.

[82] GPUmat: GPU toolbox for MATLAB.


[100] MATLAB plug-in for CUDA.


