Published in final edited form as:
J Comput Sci. 2012 September 1; 3(5): 269–279. doi:10.1016/j.jocs.2012.01.009.

An Efficient Computational Framework for the Analysis of Whole Slide Images: Application to Follicular Lymphoma Immunohistochemistry

Siddharth Samsi[a], **Ashok K. Krishnamurthy**[a], and **Metin N. Gurcan**[b]
[a]Ohio Supercomputer Center
[b]Department of Biomedical Informatics, The Ohio State University

Abstract

Follicular Lymphoma (FL) is one of the most common non-Hodgkin Lymphoma in the United States. Diagnosis and grading of FL is based on the review of histopathological tissue sections under a microscope and is influenced by human factors such as fatigue and reader bias. Computer-aided image analysis tools can help improve the accuracy of diagnosis and grading and act as another tool at the pathologist's disposal. Our group has been developing algorithms for identifying follicles in immunohistochemical images. These algorithms have been tested and validated on small images extracted from whole slide images. However, the use of these algorithms for analyzing the entire whole slide image requires significant changes to the processing methodology since the images are relatively large (on the order of 100k × 100k pixels). In this paper we discuss the challenges involved in analyzing whole slide images and propose potential computational methodologies for addressing these challenges. We discuss the use of parallel computing tools on commodity clusters and compare performance of the serial and parallel implementations of our approach.

Keywords

Follicular Lymphoma; Immunohistochemical; CD10; K-means; Parallel Computing

1. Introduction

Follicular Lymphoma (FL) accounts for 35% of all adult B cell lymphomas, and 70% of low grade lymphomas in U.S. clinical trials. A grading system adopted by the World Health Organization (WHO) divides FL into three histological grades based on the average number of centroblasts (CBs), in ten random, microscopic standard high power fields (HPF) [1]. Grade I has up to 5 CBs/HPF, grade II has 6–15 CBs/HPF and grade III has greater than 15 CBs/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. Moreover, since there is no evidence of benefit from chemotherapy for low grade FL, such patients commonly will not be treated to avoid side effects of chemotherapy. In contrast, grade III FL is an aggressive disease that is rapidly fatal if not immediately treated with aggressive chemotherapy. These differences underscore the importance of accurate histological grading of FL to appropriately risk stratify FL patients and to guide crucial clinical decisions of timing and type of chemotherapy. However, in a multi-site study, the agreement among experts for the various grades of follicular lymphoma varied between 61% and 73% [2]. Such a large disparity in FL grading underscores the need for additional diagnostic tools to increase the accuracy of FL grading. One such tool that may greatly aid pathologists in accurate FL grading is computer based image analysis.

Recent developments in imaging technology [3], [4], [5], [6] have led to the availability of high resolution slide scanners that can scan and digitize histopathology slides at magnifications up-to 40× microscope resolution. The availability of such technology has led to a significant amount of research in computer aided analysis of tissue images in order to provide a quantitative assessment of the disease [7], [8], [9]. Image analysis tools have been developed for the automated grading of neuroblastoma [10], [11], follicular lymphoma [12], [13], breast cancer [14], [15] and prostate cancer [16], [17].

Approaches to computerized analysis of FL have been studied in the past. Firestone, et al [18] used texture and frequency domain based features for the classification of follicular lymphoma. In [19], the authors used digital image analysis to study the quantification of staining in KI-67 based immunohistochemical stained FL tissue. Kaufman, et. al [20] studied the use of computer aided image analysis tools for follicle finding in H&E (Hematoxylin and Eosin) stained tissue with the conclusion that the use of computerized methods to quantify histologic features can assist pathologists in the diagnosis of follicular lymphoma. More recently, Neuma, et al [21] developed a methodology for identifying nuclei in immunohistochemically stained FL tissue section. This approach was based on color segmentation using an experimentally determined threshold, followed by an adaptive thresholding and watershed algorithm. The reported results showed an average error of 11%. These approaches have concentrated on the task of grading and classification of follicular lymphoma. Our group has been developing tools for computer-aided diagnosis of follicular lymphoma [12],[13] with promising results. The work described in [13] develops tools for the detection of follicles using H&E and immunohistochemically (IHC) stained tissue by analyzing the morphological and textural features of the images. Using these results, we developed models to describe tissue histology for classification of FL grades [12].

The use of these algorithms for analyzing the whole slide image requires significant changes to the processing methodology since the images are relatively large (on the order of 100k × 100k pixels). Therefore, while developing these methods, we have had to address the computational issues involved. Past work has included research into the use of grid computing [22] and graphics processing units (GPU) for efficient processing of histopathological images of neuroblastoma [23]. These efforts focused on specific implementations of image analysis algorithms for computer aided diagnosis. The use of GPUs for computationally intensive parts of the application typically requires the use of C/C ++. In this paper, our focus is on the use of high level languages such as MATLAB© (The MathWorks Inc., Natick, MA) which allow quick prototyping and debugging of algorithms. Kong et al [24] developed a system for the evaluation of neuroblastoma in whole slide H&E stained images through the use of a multi-resolution framework. Cambazoglu et al [22] have proposed a grid-based framework for the processing of large histopathology images with application to neuroblastoma prognosis.

This paper discusses our experience with the analysis of CD10 images using parallel computing. Lessons learned from this will be used to parallelize centroblast detection routines which are run on 40X resolution images. The focus of this paper is on the development of parallelization tools that can be used for several tasks in the computer aided diagnosis of follicular lymphoma. In this paper, we discuss the challenges involved in analyzing whole slide images and propose potential computational methodologies for addressing these challenges. We discuss the use of parallel computing tools on commodity clusters and compare performance of the serial and parallel implementations of our approach. Specifically, we describe a methodology for applying a previously developed algorithm for the detection of follicles in IHC (CD10 stained) slides [12], [25] to whole slide images. In Section 2 we briefly discuss the grading of follicular lymphoma and the need for IHC stains. Section 3 describes the proposed computer aided system for the detection of follicles in FL slides along with the computational aspects of the algorithm used for follicle segmentation. We highlight the challenges encountered in the processing of histopathological images using existing software tools and present potential solutions. Section 4 explains the serial processing of whole slides and the implementation of a parallel algorithm for processing whole slide images in Section 5. Finally, we present results and observations in Section 6.

2. Follicular Lymphoma Grading

Follicular Lymphoma grading is based on the number of centroblasts in ten representative follicle regions in H&E stained tissues. H&E stained tissue is used for counting centroblasts because the stain makes it possible to identify cellular details such as the nuclei and cytoplasm in the tissue. However, identification of follicles is not always easy in H&E stained images. In such cases, immunohistochemical (IHC) stained tissues can be used to identify follicles. IHC stains are used to mark the tissue in hues of brown and blue color. Follicles, which consist mostly of B-cells are stained dark brown and can be identified with relative ease. One such example is shown in Figure 1, where follicle identification is challenging in the H&E stained tissue (Figure 1(a)) and significantly easier in the IHC stained tissue shown in Figure 1(b) and 1(c) respectively. The IHC stains typically used in this task are CD10 and CD20. Samples of CD10 tissue sections at 4× resolution are shown in Figure 2. In both images, the blue borders identify the follicle regions. These figures also illustrate the wide variations in size and shape of the follicles, which makes it harder to model them using well-defined geometric entities such as circles or ellipses.

In this study we have used twelve CD10 stained tissues to test the performance of our follicle segmentation algorithm. Images varied in size and a summary is provided in Table 3.

3. System For Follicle Detection

We can broadly split the task of follicular lymphoma grading into two main procedures: (1) Identification of follicles; and (2) Counting centroblasts in the follicles. This paper focuses on the task of follicle segmentation in IHC stained whole slide images of follicular lymphoma. Images of IHC stained tissues used in this research were scanned using an Aperio ScanScope (Aperio, San Diego, CA) slide digitizer at 40× microscope magnification. Depending on the tissue itself, scanned images can be as large as 100,000 × 100,000 in pixel dimensions. For example, one of the images used in this study was 96,899 × 174,600 pixels in size.

3.1. Identification of Follicle Regions

In [25], we propose an algorithm for identification of follicle regions in CD10 images. Figure 3 shows the flowchart of the proposed algorithm. The proposed algorithm involves calculation of color and grayscale features which are used as feature vectors for k-means clustering. The color feature used is the hue channel from the HSV (Hue-Saturation-Value) colorspace conversion of the original image. The first texture feature used is the output of a median filter of size 45×45 applied to a grayscale version of the image. The second texture feature is the energy feature calculated from the co-occurrence matrix as defined by Haralick [26]. The output of the clustering algorithm provides a segmented image that is further processed by an iterative watershed algorithm followed by a boundary smoothing step. For a detailed description of the algorithm, readers are referred to [25]. The images used in this analysis were scaled down from the original 40× resolution to 4× resolution for processing using the Aperio ImageScope software. A resolution of 4× was chosen because it was found to be sufficient for the task of isolating follicles. If the structures inside the follicles are to be analyzed, a higher resolution image will be needed.

3.2. Computation and Memory Requirements of Proposed System

The follicle detection algorithm described in this paper was applied to an image of size 9,690 × 17,460 pixels. This image is at 4× resolution and was scaled down from the original size of 96,899 × 174,600 pixels. The algorithm was implemented in MATLAB© because of the ease in prototyping and debugging. The two main challenges to implementing the serial version of our algorithm were:

- Physical limits - Large amounts of memory is required to process the whole slide image as described in Section 3.2.1. Even at 4× resolution, the images can be too large to be processed as a single array. By employing a block-processing methodology, we analyze whole slide images serially as described in Section 4.1.
- Computation time - Given sufficient amount of memory, the total computation time can still be excessive, thus making the automated analysis impractical. Analysis of the 4× image of size 9,690 × 17,460 pixels took one hour.

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 1 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 1, the "Block artifact removal" basically involves application of the k-means algorithm to a specific region in the image, as described in Section 4.1.1.

3.2.1. Memory Usage for Algorithm—In order to process an image of size 9,690 × 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 features used for clustering requires approximately 7.14 GB of memory. A summary of this memory usage is presented in Table 2. 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 step 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 8× or 16×) imposes even greater requirements on the system. Thus, processing the entire image as a single array is impractical and alternative methodologies are needed.

4. Whole Slide Processing

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 3.2. 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.

4.1. Overcoming Memory Limits

The problem of memory limits described in Section 3.2.1 can be solved by simply processing the entire image in blocks of size B_wxB_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 an B_wxB_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*h*8 bytes of memory. Using the earlier image size of 9,690 × 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 brings in additional challenges. The first challenge is the choice of the block size used for processing the image. Block sizes 256×256, 512×512, 1024×1024 and 2048×2048 were tested. We chose 512 × 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 due to the fact that 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 illustrates this problem. The green lines in Figure 4(a) represent the boundaries of the 512 × 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 thus need to be re-processed.

4.1.1. Identifying and Removing Block Artifact—In order to remove the block artifacts, we propose a procedure that uses morphological reconstruction [27] to first identify those objects that touch the tile borders. This is achieved by first generating a binary image with a grid with spacing equal to the block size used for processing the image. The green lines in Figure 4(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(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(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(a) using the binary image shown in Figure 4(c) as the marker results in an image that only contains the objects that require further processing. These objects are shown in Figure 4(d).

Thus, referring to Figure 4(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.

4.2. 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 [28], [29], [30] 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™.
- **pMATLAB:** The pMATLAB [31] 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) [31], [32] semantics. The underlying mechanism for communication is the MatlabMPI [33] library.
- **bcMPI:** The bcMPI [34] library is an open source toolbox developed at the Ohio Supercomputer Center. It is intended as an alternative to MatlabMPI and mainly targeted towards large, shared supercomputers. The bcMPI library uses OpenMPI to provide MPI [35] style message passing over the high speed network on a cluster, thus giving the user control over fine grained communication.

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, thus enabling one to work on significantly larger data sizes as compared to a single machine and also provides tools for message passing between multiple processes. [36].

5. Parallel Implementation

5.1. Parallel 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 that works on fixed sized windows can be run in a parallel fashion using our parallel filter code as described in Section 5.2. The K-means clustering was parallelized by simply using serial implementations of K-means on distributed matrices. Figure 5 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.

5.2. 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 × 45 windows and the texture calculations are performed on 15 × 15 windows for 4× 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 -

- **i.** Distribute 2D image matrix across Np 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.
- **ii.** All processors exchange $(m-1)/2$ padding columns of data with their neighbors - This is illustrated in Figure 6 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 data to be filtered by each processor.
- **iii.** Apply filter to local data on each processor - The 2D filter is applied to the padded matrix on each processor. In Figure 6, 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.
- **iv.** 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 it's neighbors, there is some communication overhead that 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.

5.3. K-means On Distributed Matrices

The K-means clustering algorithm is a well studied approach to data clustering [37], [38], [39]. Several parallel implementations of the algorithms have been developed [40], [41], [42] including implementations that run on graphics processing units (GPUs) [43], [44]. We have used the k-means++[45] 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 4.1, with the block size being 512 × 512 pixels.

6. 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 40× microscope resolution. All images were originally in the SVS image format developed by Aperio and is a JPEG-2000 compressed TIFF format. Images were down-sampled to 4× resolution and converted to uncompressed TIFF format using the ImageScope [46] 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 4× resolution are shown in Table 3.

The serial and parallel implementations of our algorithm were applied to whole slides images of follicular lymphoma. The analysis was done on the high performance computing resources (HPC) at the Ohio Supercomputer Center [47]. The cluster is an AMD Opteron based system consisting of 877 dual socket, dual core 2.6 GHz 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 4.1. Finally, the parallelized version of our algorithm was run on the dual socket, quad core AMD Opteron systems 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 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 4.1.1. 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 3 and 4. However, the use of block processing enables the processing of larger images that will require significantly greater amounts on memory for processing.

6.1. Parallel algorithm performance

From Table 4 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.

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 × 17459 (Image 1 in Table 3) 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 5.

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 7 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 8 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 9 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 10. The average time was calculated across all images and all processor combinations. The timing clearly shows a trend of decreasing time required to process whole slide images as the number of processors is increased.

7. Conclusion

In this paper 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 quickly prototype such algorithms and test on whole slides using parallel computing can be a powerful tools towards the development of automated computer-aided diagnosis systems. 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.

Acknowledgments

The project described was supported in part by Award Number R01CA134451 from the National Cancer Institute.

The authors would like to thank Dr. Gerard Lozanski of the Ohio State University, Department of Pathology for providing the tissue slides and for his expert guidance on identifying follicles in IHC images.

References

1. Jaffe, ES.; Harris, NL.; Stein, H.; Vardiman, JW. World health organization classification of tumours of haematopoietic and lymphoid tissues. IARC Press; 2001.
2. The Non-Hodgkinś Lymphoma Classification Project. A clinical evaluation of the international lymphoma study group classification of non-hodgkin’s lymphoma. Blood. 1997; 89(11):3909–3918. [PubMed: 9166827]
3. Aperio Inc. 2011. http://www.aperio.com
4. 3DHISTECH Ltd. 2011. http://www.3dhistech.com
5. CRI Inc. 2011. http://www.cri-inc.com
6. BioImagene. 2011. http://www.bioimagene.com
7. Gurcan MN, Boucheron LE, Can A, Madabhushi A, Rajpoot NM, Yener B. Histopathological image analysis: A review. Biomedical Engineering, IEEE Reviews. 2009; 2:147–171.
8. Peng Y, Jiang Y, Chuang ST, Yang XJ. Computer-aided detection of prostate cancer on tissue sections. Applied Immunohistochemistry & Molecular Morphology. 2009; 17(5)
9. Madabhushi, A.; Basavanhally, A.; Doyle, S.; Agner, S.; Lee, G. Computer-aided prognosis: Predicting patient and disease outcome via multi-modal image analysis; Biomedical Imaging: From Nano to Macro, 2010 IEEE International Symposium on; 2010. p. 1415-1418.
10. Gurcan, MN.; Kong, J.; Sertel, O.; Cambazoglu, BB.; Saltz, J.; Catalyurek, U. Computerized pathological image analysis for neuroblastoma prognosis; AMIA Annual Symposium. American Medical Informatics Association; 2007. p. 304-308.

11. Sertel O, Kong J, Shimada H, Catalyurek U, Saltz J, Gurcan M. Computer-aided prognosis of neuroblastoma on whole-slide images: Classification of stromal development. Pattern Recognition. 2009; 42(6):1093–1103. [PubMed: 20161324]

12. Sertel O, Catalyurek UV, Lozanski G, Saltz JH, Gurcan MN. Histopathological image analysis using model-based intermediate representations and color texture: Follicular lymphoma grading. The Journal of Signal Processing Systems for Signal, Image, and Video Technology. 2008; 55(1): 169–183.

13. Sertel O, Kong J, Lozanski G, Catalyurek U, Saltz JH, Gurcan MN. Computerized microscopic image analysis of follicular lymphoma. Medical Imaging 2008: Computer-Aided Diagnosis. 2008; 6915(1):691535. (pages 11).

14. Schnorrenberg F, Pattichis CS, Kyriacou KC, Schizas CN. Computer-aided detection of breast cancer nuclei. Information Technology in Biomedicine, IEEE Transactions on. 1997; 1(2):128–140.

15. Petushi S, Katsinis C, Coward C, Garcia F, Tozeren A. Automated identification of microstructures on histology slides. Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on. 2004; Vol. 1:424–427.

16. Naik, S.; Doyle, S.; Agner, S.; Madabhushi, A.; Feldman, M.; Tomaszewski, J. Automated gland and nuclei segmentation for grading of prostate and breast cancer histopathology; Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on; 2008. p. 284-287.

17. Teverovskiy M, Kumar V, Ma J, Kotsianti A, Verbel D, Tabesh A, et al. Improved prediction of prostate cancer recurrence based on an automated tissue image analysis system. Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on. 2004; Vol. 1:257–260.

18. Firestone L, Preston K, Nathwani BN. Continuous class pattern recognition for pathology, with applications to non-hodgkin's follicular lymphomas. Pattern Recognition. 1996; 29(12):2061–2078.

19. Schwartz B, Pinkus G, Bacus S, Toder M, Weinberg D. Cell proliferation in non-Hodgkin's lymphomas. Digital image analysis of Ki-67 antibody staining. The American journal of pathology. 1989; 134(2):327–336. [PubMed: 2464940]

20. Kaufman AG, Nathwani BN, Preston K Jr. Subclassification of follicular lymphomas by computerized microscopy. Human Pathology. 1987; 18(3):226–231. [PubMed: 3817804]

21. Neuman, U.; Korzynska, A.; Lopez, C.; Lejeune, M. Segmentation of stained lymphoma tissue section images. In: Pietka, E.; Kawa, J., editors. Information Technologies in Biomedicine; vol. 69 of Advances in Soft Computing. Heidelberg: Springer Berlin; 2010. p. 101-113.

22. Cambazoglu, BB.; Sertel, O.; Kong, J.; Saltz, J.; Gurcan, MN.; Catalyurek, UV. Efficient processing of pathological images using the grid: computer-aided prognosis of neuroblastoma. Proceedings of the 5th IEEE workshop on Challenges of large applications in distributed environments; ACM; 2007. p. 35-41.

23. Ruiz A, Sertel O, Ujaldon M, Catalyurek U, Saltz J, Gurcan M. Stroma classification for neuroblastoma on graphics processors. International journal of data mining and bioinformatics. 2009; 3(3):280–298. [PubMed: 19623771]

24. Kong J, Sertel O, Shimada H, Boyer KL, Saltz JH, Gurcan MN. Computer-aided evaluation of neuroblastoma on whole-slide histology images: Classifying grade of neuroblastic differentiation. Pattern Recognition. 2009; 42(6):1080–1092. Digital Image Processing and Pattern Recognition Techniques for the Detection of Cancer.

25. Samsi S, Lozanski G, Shanarah A, Krishanmurthy AK, Gurcan MN. Detection of follicles from ihc-stained slides of follicular lymphoma using iterative watershed. Biomedical Engineering, IEEE Transactions on. 2010; 57(10):2609–2612.

26. Haralick RM, Shanmugam K, Dinstein I. Textural features for image classification. Systems, Man and Cybernetics, IEEE Transactions on. 1973; 3(6):610–621.

27. Vincent L. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. Image Processing, IEEE Transactions on. 1993; 2(2):176–201.

28. Krishnamurthy A, Nehrbass J, Chaves J, Samsi S. Survey of Parallel MATLAB Techniques and Applications to Signal and Image Processing. IEEE International Conference on Acoustics, Speech and Signal Processing. 2007; vol. 4:IV–1181–IV–1184. 2007.

29. Samsi S, Gadepally V, Krishnamurthy A. Matlab for signal processing on multiprocessors and multicores. Signal Processing Magazine, IEEE. 2010; 27(2):40–49.

30. Choy R, Edelman A. Parallel matlab: Doing it right. Proceedings of the IEEE. 2005; 93(2):331–341.

31. Bliss NT, Kepner J. pMATLAB parallel MATLAB library. International Journal of High Performance Computing Applications. 2007; 21:336–359.

32. Yelick, K.; Bonachea, D.; Chen, WY.; Colella, P.; Datta, K.; Duell, J., et al. PASCO ’07: Proceedings of the 2007 international workshop on Parallel symbolic computation. New York, NY, USA: ACM; 2007. Productivity and performance using partitioned global address space languages; p. 24-32.ISBN 978-1-59593-741-4; doi: http://doi.acm.org/10.1145/1278177.1278183

33. Kepner J, Ahalt S. MatlabMPI. Journal of Parallel and Distributed Computing. 2004; 64(8):997–1005.

34. Hudak, D.; Ludban, N.; Gadepally, V.; Krishnamurthy, A. Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing Applications. IEEE Computer Society; 2007. Developing a computational science IDE for HPC systems; p. 5

35. Forum MPI. MPI: A Message-Passing Interface. Version 2.2. 2009

36. Sharma G, Martin J. Matlab®: a language for parallel computing. International Journal of Parallel Programming. 2009; 37(1):3–36.

37. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY. An efficient k-means clustering algorithm: analysis and implementation. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 2002; 24(7):881–892.

38. Wilkin, GA.; Huang, X. K-means clustering algorithms: Implementation and comparison; Computer and Computational Sciences, 2007. IM-SCCS 2007. Second International Multi-Symposiums on; 2007. p. 133-136.

39. Peters G. Some refinements of rough k-means clustering. Pattern Recogn. 2006; 39:1481–1491.

40. Stoffel, K.; Belkoniene, A. Parallel k/h&-means clustering for large data sets. In: Amestoy, P.; Berger, P.; Daydé, M.; Ruiz, D.; Duff, I.; Frayssé, V., editors. Euro-Par ’99 Parallel Processing; vol. 1685 of Lecture Notes in Computer Science. Heidelberg: Springer Berlin; 1999. p. 1451-1454.

41. Kerdprasop, K.; Kerdprasop, N. Proceedings of the 10th WSEAS international conference on Applied computer science. ACS’10. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS); 2010. Parallelization of k-means clustering on multi-core processors; p. 472-477.ISBN 978-960-474-231-8

42. Kraj P, Sharma A, Garge N, Podolsky R, McIndoe R. Parakmeans: Implementation of a parallelized k-means algorithm suitable for general laboratory use. BMC Bioinformatics. 2008; 9(1):200–213. [PubMed: 18416829]

43. Zechner, M.; Granitzer, M. Accelerating k-means on the graphics processor via cuda; Intensive Applications and Services, 2009. INTENSIVE ’09. First International Conference on; 2009. p. 7-15.

44. Wu, R.; Zhang, B.; Hsu, M. Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop. UCHPC-MAW 09. New York, NY, USA: ACM; 2009. Clustering billions of data points using gpus; p. 1-6.ISBN 978-1-60558-557-4

45. Arthur, D.; Vassilvitskii, S. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. SODA ’07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics; 2007. k-means++: the advantages of careful seeding; p. 1027-1035.ISBN 978-0-898716-24

46. Aperio Inc. 2011. http://www.aperio.com/download-imagescope-viewer.asp

47. Ohio Supercomputer Center. 2011. http://www.osc.edu

Biographies

Siddharth Samsi received his Bachelor's degree in Electronics & Telecommunications from the College of Engineering, Pune, India and Masters degree in Electrical Engineering from The Ohio State University. He is currently a Ph.D. student in the Electrical and Computer Engineering Department at The Ohio State University. He joined the Ohio Supercomputer Center in 2006 and conducts research in parallel computing using High Level Languages and image processing for medical applications.

Ashok Krishnamurthy earned his bachelor's degree in Electrical Engineering from the Indian Institute of Technology in Madras, India, in 1979. He received his master's degree and doctorate in electrical engineering at the University of Florida in 1981 and 1983, respectively. He is the Director of Research and acting co-Director at the Ohio Supercomputer Center and also serves as an Associate Professor in the Electrical and Computer Engineering department at The Ohio State University. He conducts research in signal/image processing, high performance computing, parallel high-level language applications and computational models of hearing.

Metin N. Gurcan is an Associate Professor at the Department of Biomedical Informatics at The Ohio State University. He received his M.Sc. degree in Digital Systems Engineering from UMIST, England and his B.Sc. and Ph.D. degrees in Electrical and Electronics Engineering from Bilkent University, Turkey. He was a postdoctoral research fellow and later a research investigator in the Department of Radiology at the University of Michigan, Ann Arbor. Prior to joining The Ohio State University in October 2005, he worked as a senior researcher and product director at a high-tech company, specializing in computer-aided detection and diagnosis of cancer from radiological images. Dr. Gurcan's research interests include image analysis and understanding, computer vision with applications to medicine. He is the author of over 75 peer-reviewed publications and has two patents in computer-aided diagnosis in volumetric imagery. Dr. Gurcan is a senior member of IEEE, RSNA, SPIE.

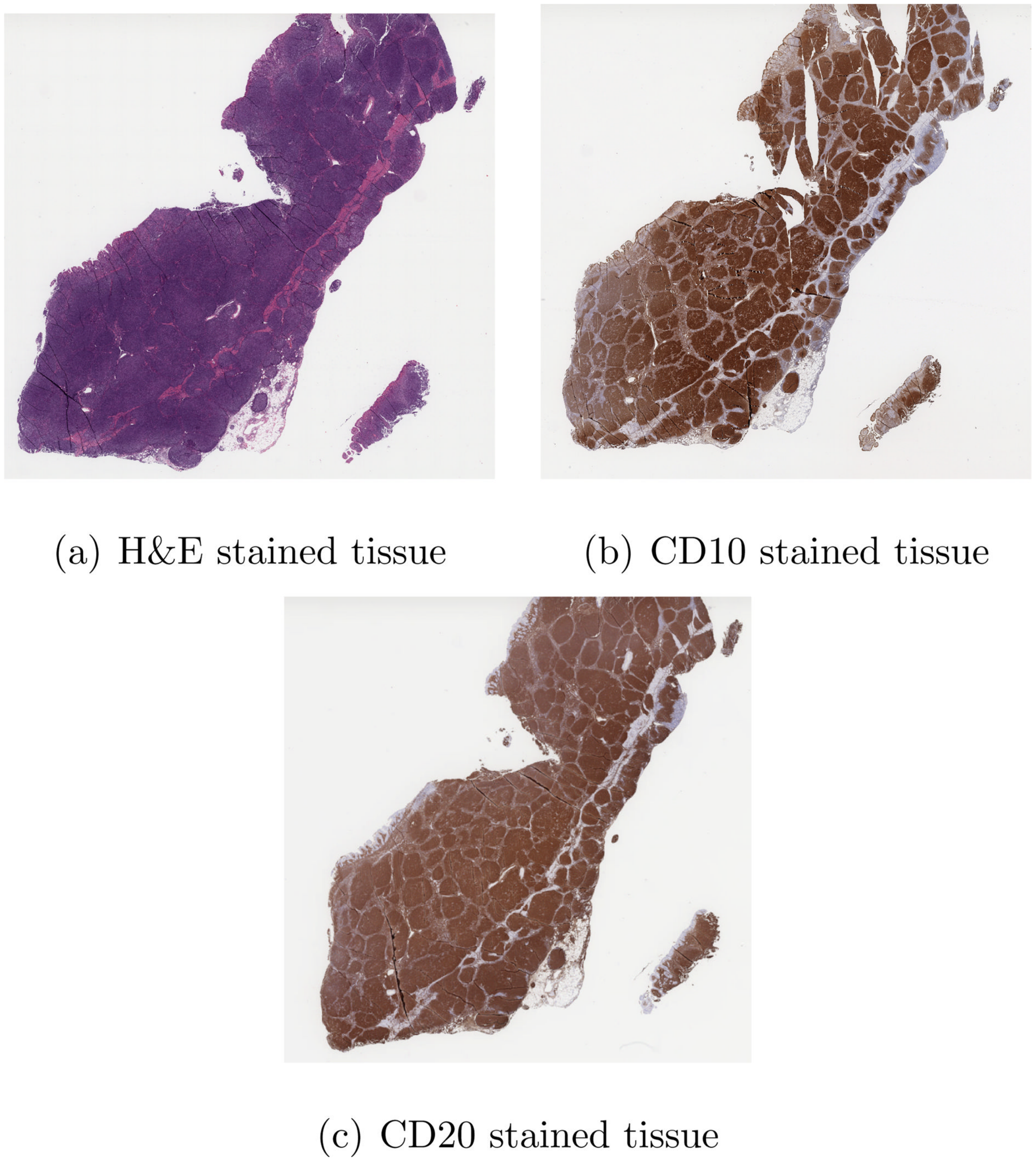


Figure 1.

H&E (1(a)) and IHC (1(b), 1(c)) stained tissue: Adjacent tissue slices stained with H&E and different IHC stains used to visualize different tissue components

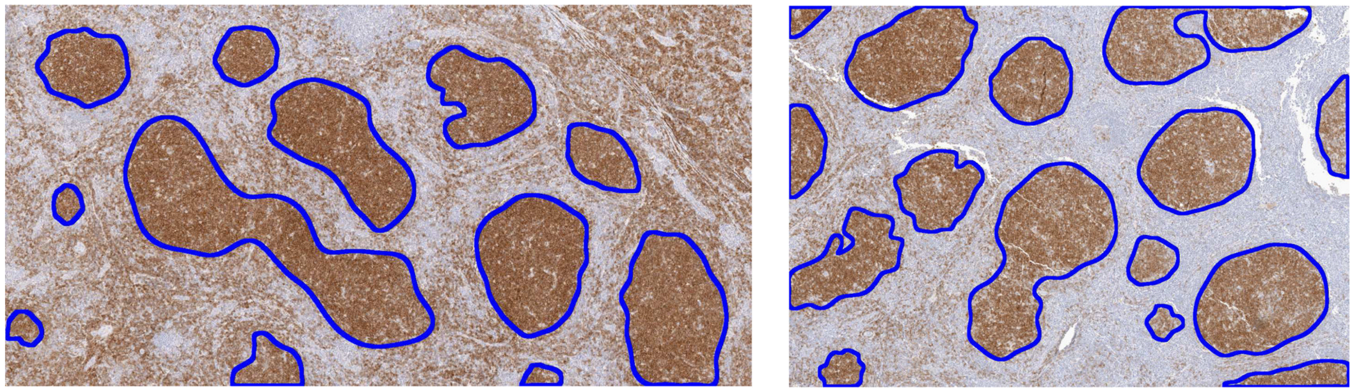


Figure 2.
Follicles in CD10 stained tissue shown at 4× microscope resolution

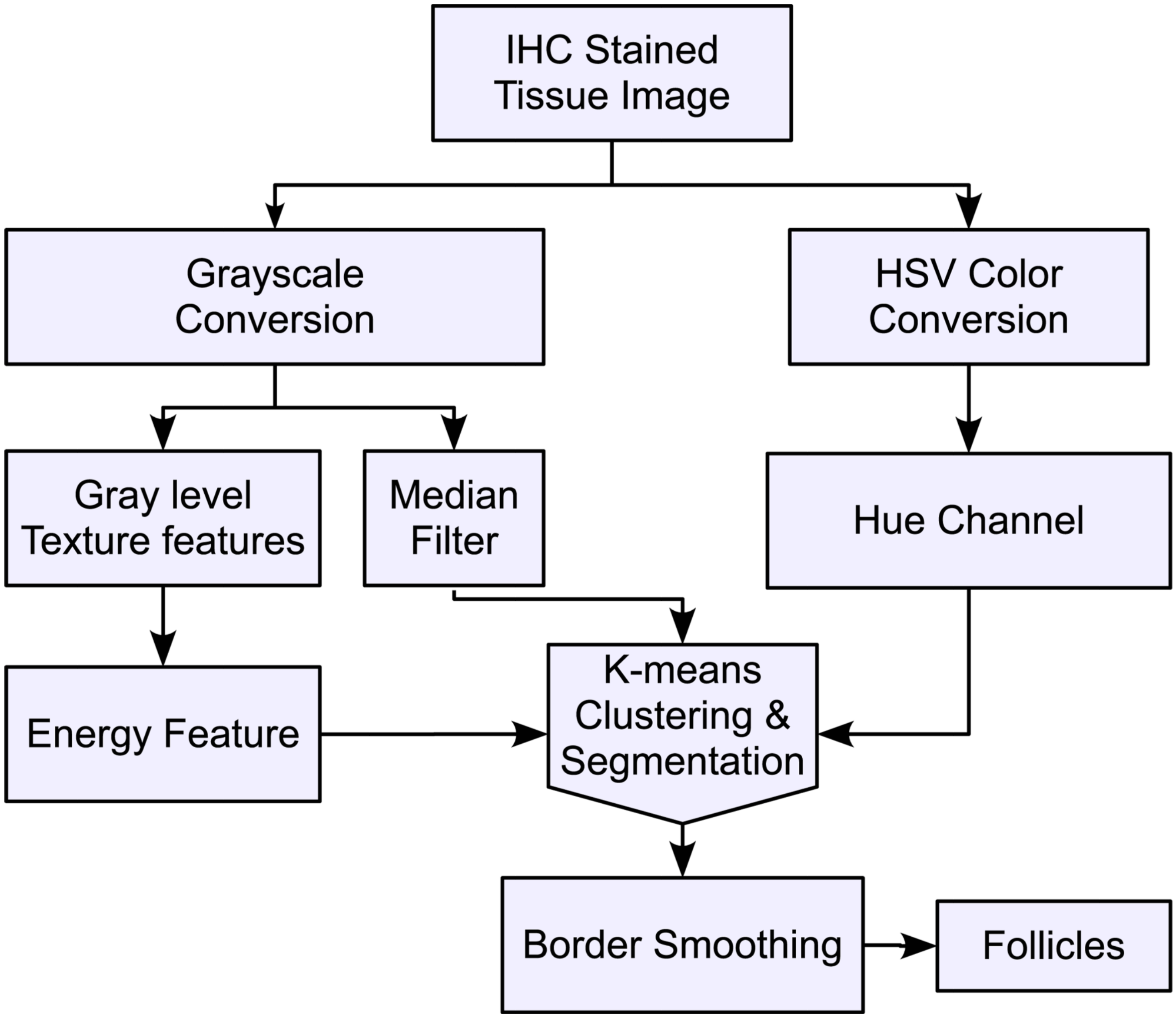
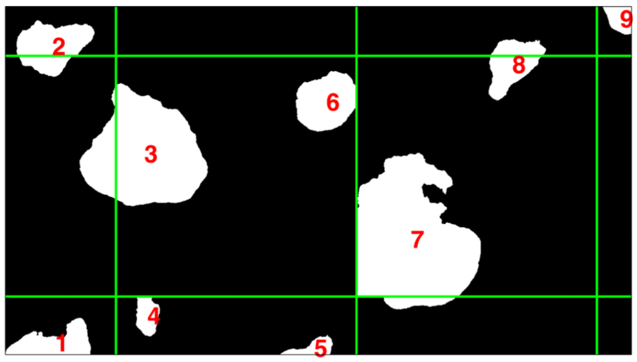
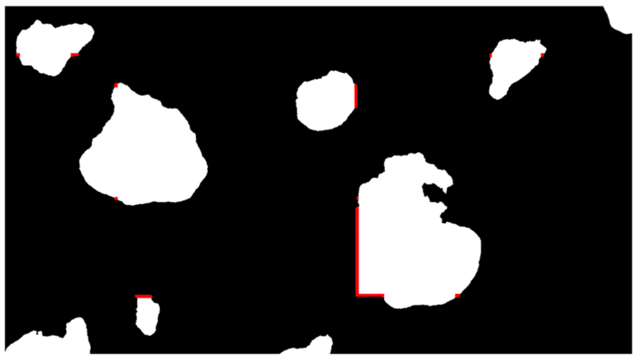


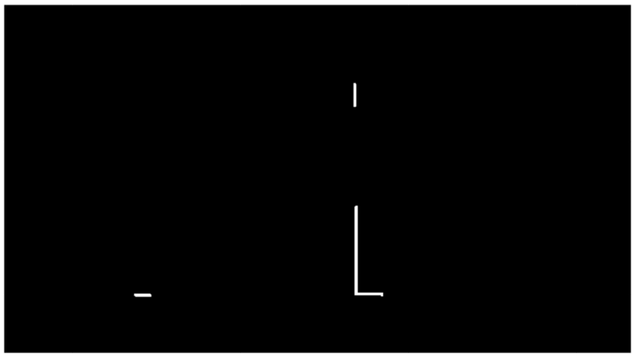
Figure 3.
Flowchart for Follicle Detection in CD10 images



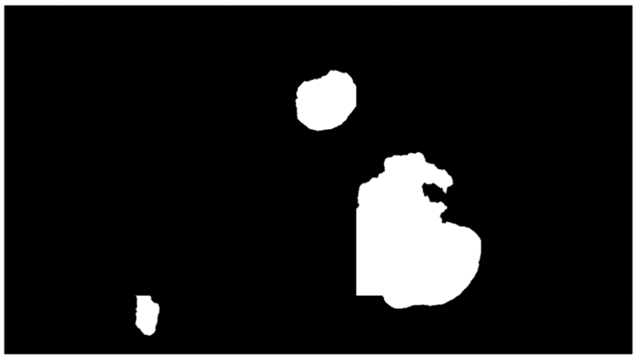
(a) Grid line superimposed on identified objects



(b) Red lines indicate intersection of grid lines and object borders



(c) Marker image used for identifying objects requiring post-processing



(d) Objects requiring post-processing

Figure 4.
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

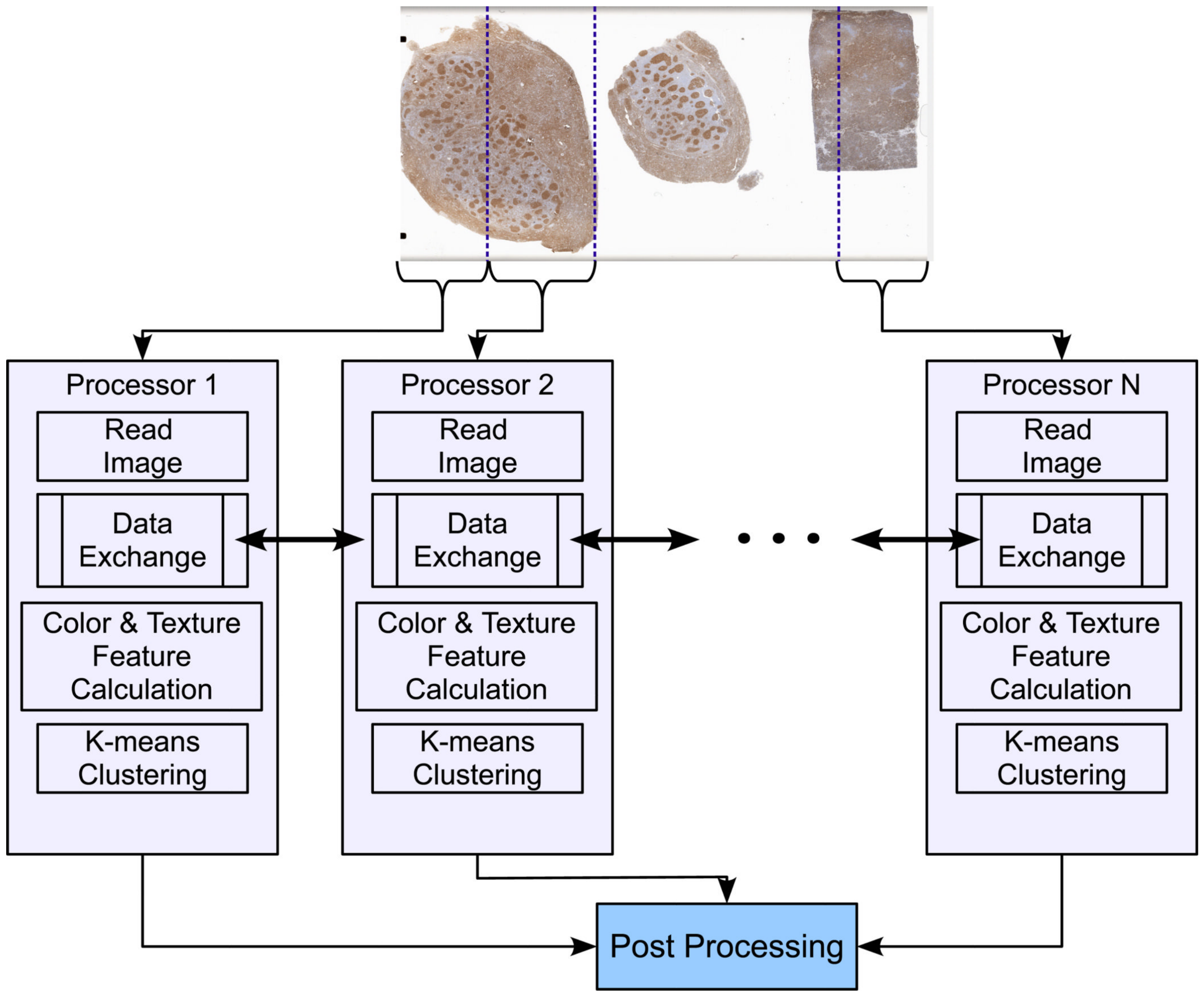


Figure 5.
Parallel implementation: Image is distributed across N_p processors and processed in $B_w \times B_h$ blocks on each processor

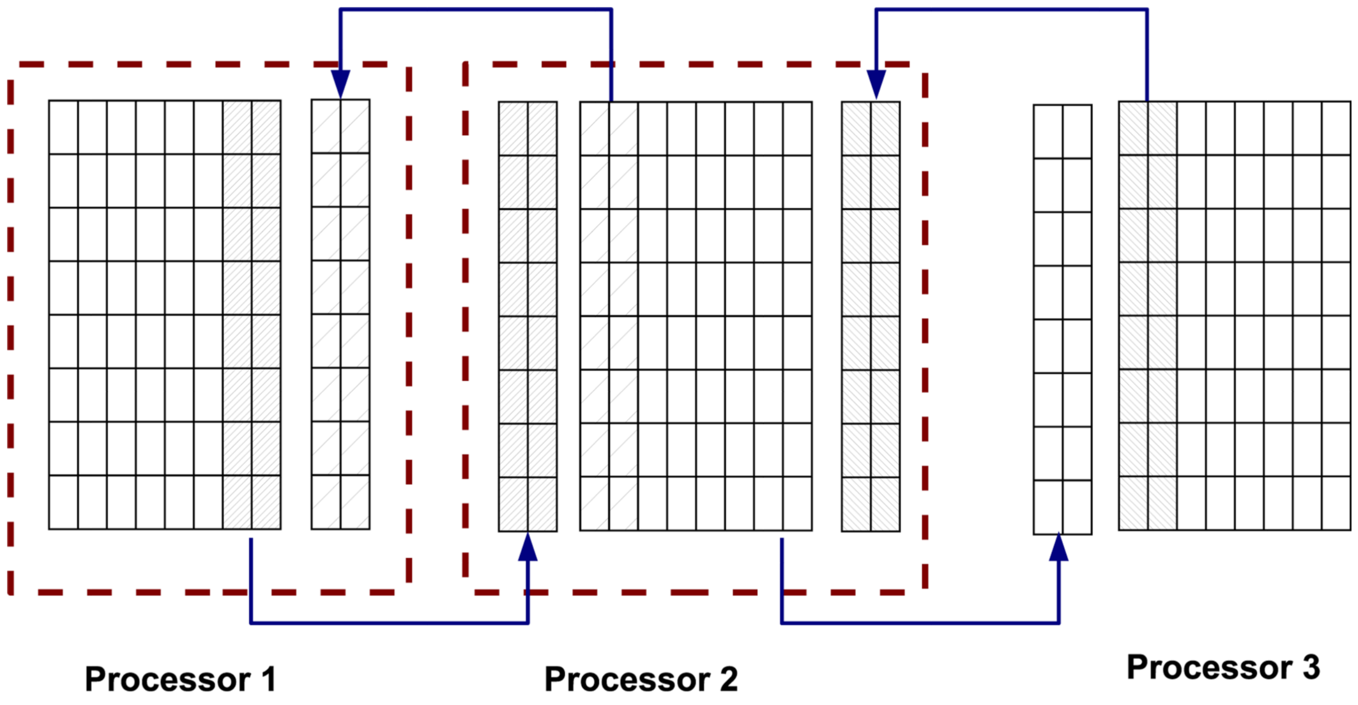


Figure 6.
Inter-processor communication: Data exchange between processors is indicated by the blue lines. Processors need to exchange borders columns/rows of data

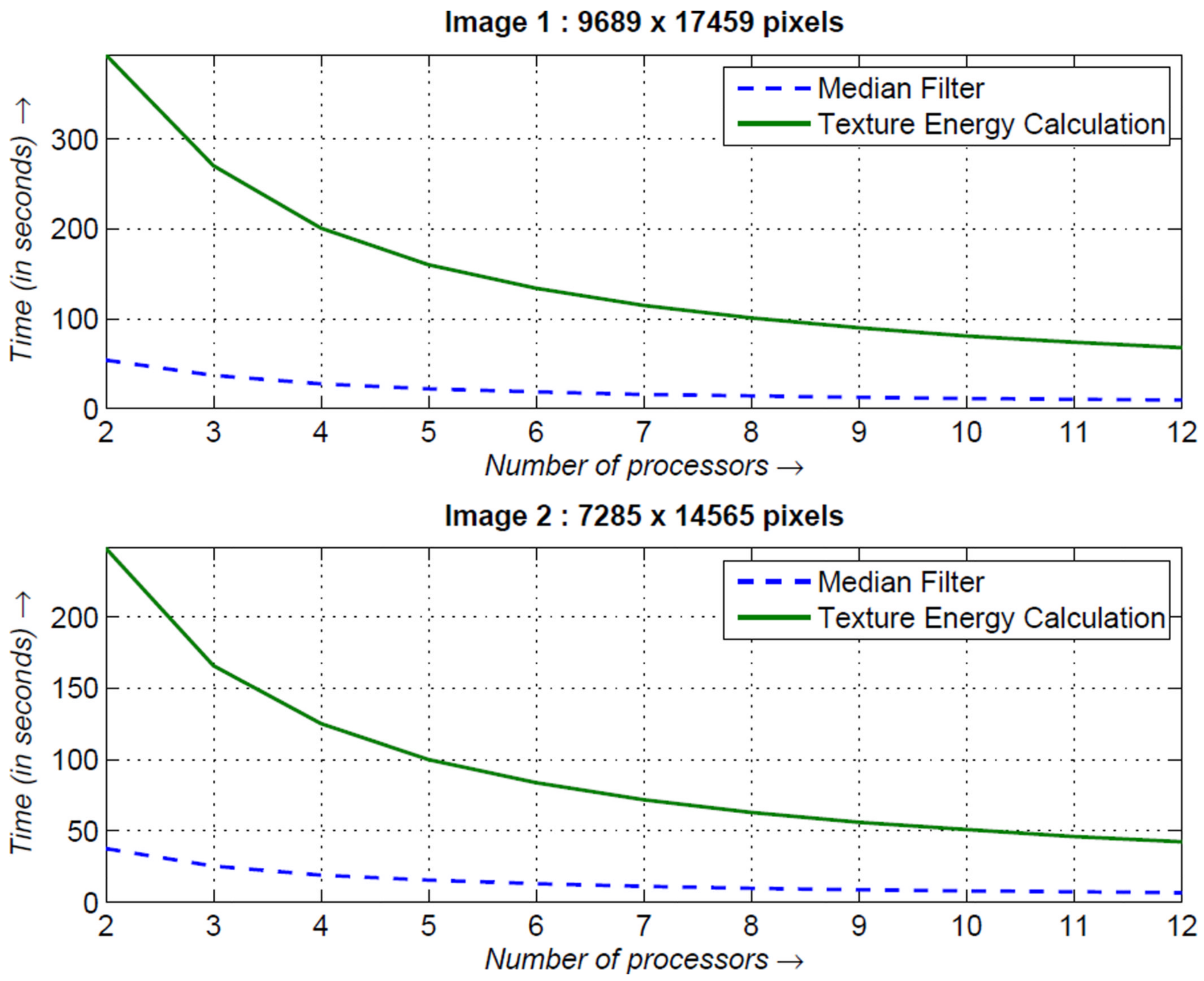


Figure 7.
Run time for parallel median filter and texture energy calculation for two images used in the study

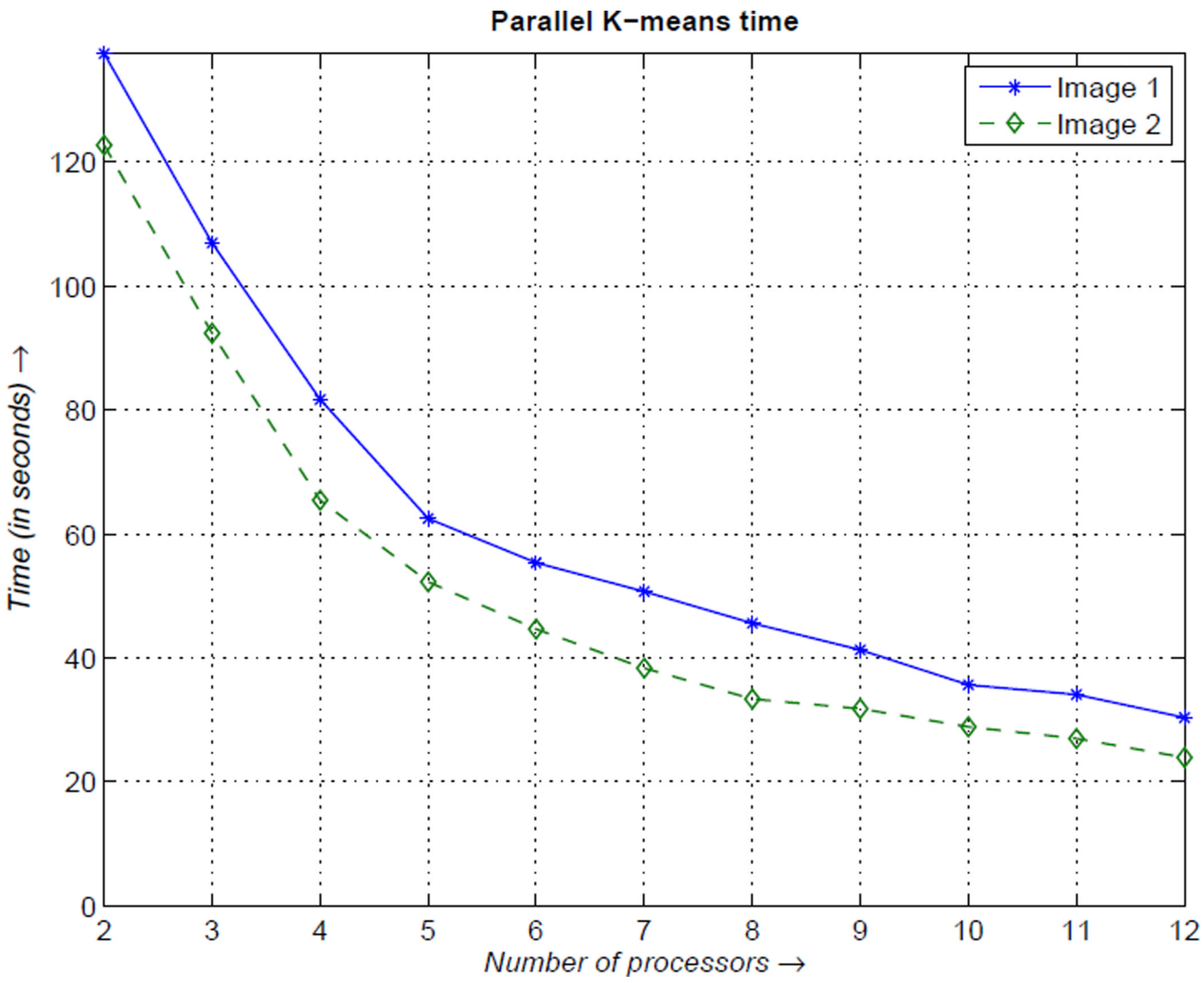


Figure 8.
Run time for parallel k-means calculations for two images used in the study

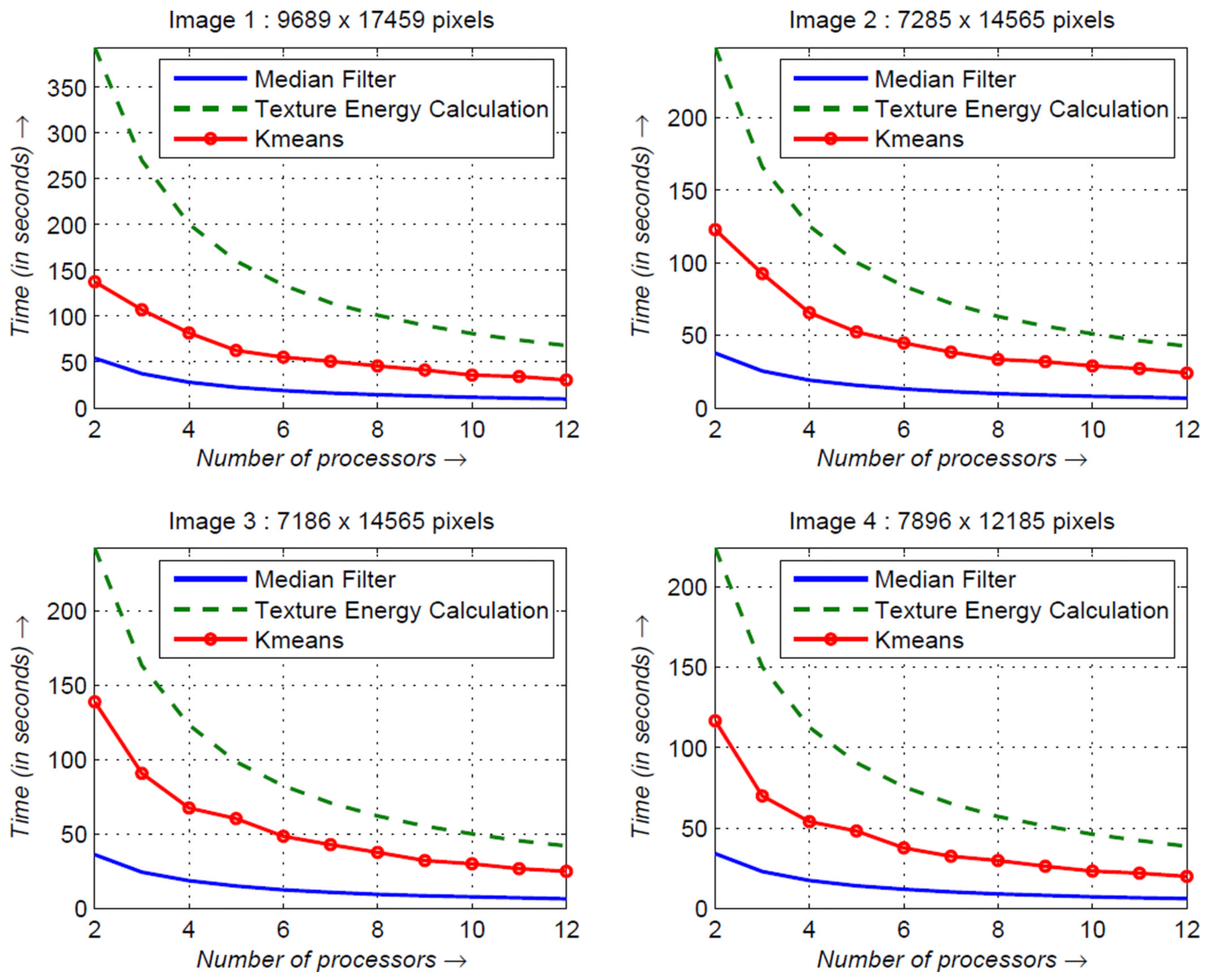


Figure 9.
Average timing for parallel median filtering, texture energy calculation and k-means clustering for four CD10 stained tissue images used in the study

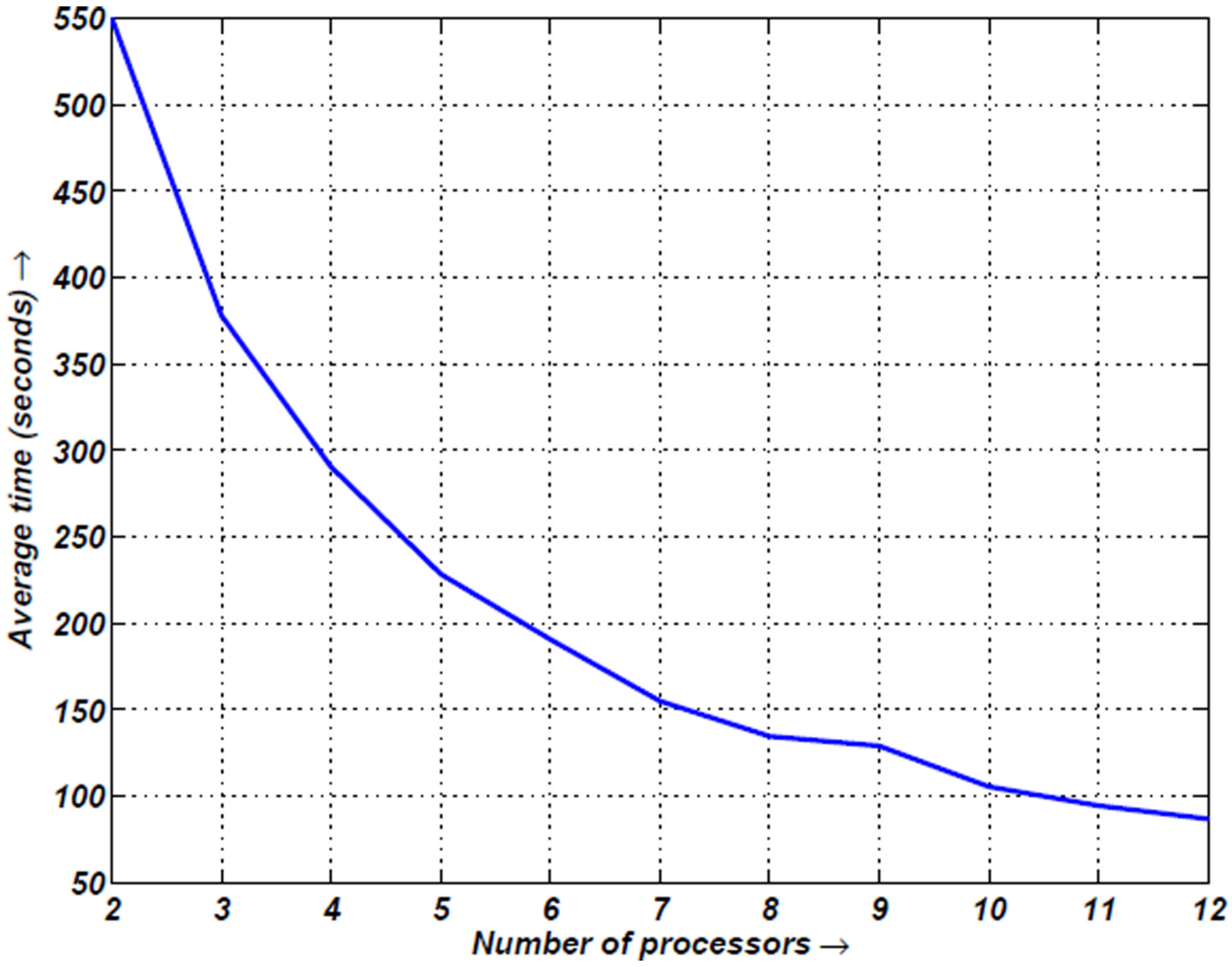


Figure 10.

Computation time for parallel algorithm: Average time for entire algorithm calculated over 10 runs of the algorithm across all images used in the study

Table 1

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 3

Operation	**Percent of total time**
Median Filter	0.974 %
Texture calculation	56.722 %
K-means Clustering	16.33 %
Block artifact removal	17.141 %
Other operations	8.832 %

Table 2

Memory usage for serial implementation when analyzing whole slide image of size 9,690 × 17,460 pixels

Data	Memory Used
RGB Image	507 MB
HSV Image	4 GB
Texture energy feature	1.3 GB
Median filter output	1.3 GB

Table 3

Image sizes for data used in this study

Image	**Size (in pixels)**
Image 1	9689 × 17459
Image 2	7285 × 14565
Image 3	7186 × 14565
Image 4	7896 × 12185
Image 5	8657 × 10567
Image 6	8425 × 10662
Image 7	8859 × 9797
Image 8	7977 × 9234
Image 9	6910 × 9125
Image 10	7809 × 7491
Image 11	7064 × 8259
Image 12	7168 × 7396

Table 4

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

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

Table 5

Data distribution across ten processors

Processor Number	Non-background pixels (in millions)	Number of calculations for 4 centers (in millions)
1	11.4467	45.8670
2	15.2282	60.9128
3	14.9236	59.6944
4	8.5181	34.0725
5	8.0459	32.1836
6	10.8125	43.2502
7	4.9670	19.8682
8	4.1810	16.7242
9	11.4379	45.7516
10	10.3697	41.4788