

Symposium - Original Research

Graphical processing unit implementation of an integrated shape-based active contour: Application to digital pathology

Sahirzeeshan Ali, Anant Madabhushi¹

Department of Electrical and Computer Engineering, ¹Department of Biomedical Engineering, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA

E-mail: *Sahirzeeshan Ali - sahirali@eden.rutgers.edu

*Corresponding author

Received: 20 October 11

Accepted: 20 October 11

Published: 19 January 12

This article may be cited as:

Ali S, Madabhushi A. Graphical processing unit implementation of an integrated shape-based active contour: Application to digital pathology. J Pathol Inform 2011;2:S13.

Available FREE in open access from: <http://www.jpathinformatics.org/text.asp?2011/2/2/13/92029>

Copyright: © 2011 Ali S. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

Commodity graphics hardware has become a cost-effective parallel platform to solve many general computational problems. In medical imaging and more so in digital pathology, segmentation of multiple structures on high-resolution images, is often a complex and computationally expensive task. Shape-based level set segmentation has recently emerged as a natural solution to segmenting overlapping and occluded objects. However the flexibility of the level set method has traditionally resulted in long computation times and therefore might have limited clinical utility. The processing times even for moderately sized images could run into several hours of computation time. Hence there is a clear need to accelerate these segmentations schemes. In this paper, we present a parallel implementation of a computationally heavy segmentation scheme on a graphical processing unit (GPU). The segmentation scheme incorporates level sets with shape priors to segment multiple overlapping nuclei from very large digital pathology images. We report a speedup of 19x compared to multithreaded C and MATLAB-based implementations of the same scheme, albeit with slight reduction in accuracy. Our GPU-based segmentation scheme was rigorously and quantitatively evaluated for the problem of nuclei segmentation and overlap resolution on digitized histopathology images corresponding to breast and prostate biopsy tissue specimens.

Key words: GPU Implementation, Parallel Processing, Level set, Medical imaging, Segmentation, Digital Pathology, Histopathology, Fast Active Contour, Multi-threaded programming

Access this article online

Website:

www.jpathinformatics.org

DOI: 10.4103/2153-3539.92029

Quick Response Code:



INTRODUCTION

In the rapidly developing field of digital pathology,^[1] ability to accurately segment structures on digitized histology images, especially the ability to deal with overlapping and occluded structures, is highly critical in the context of a number of different diagnostic and prognostic applications.^[2,3] In the context of prostate cancer (CaP) and breast cancer (BC), digitized

hematoxylin and eosin (H and E)-stained tissue samples are often characterized by the presence of densely packed nuclei, in which overlap between two or more nuclei has proven to be a significant challenge to computerized segmentation schemes.^[1-3] Shape-based active contours (level sets) have emerged as one of the natural solutions to overlap resolution. Recently, Ali *et al.*^[3] proposed a shape-based hybrid active contour model that can segment all of the overlapping and nonoverlapping nuclei

simultaneously in an image.

While level sets have found wide applicability for medical image segmentation, their clinical utility has been limited by (1) the amount of time needed for computation and (2) a number of free parameters that must be fine tuned for each specific application. These two limitations are intertwined, whereby users find it impractical to explore the entire space of possible parameter settings when an example result from a point in that space requires minutes or hours to generate. We have previously addressed the latter issue by automating the parameter selection associated with initialization.^[3]

Higher level segmentation approaches, such as deformable models, have been ported to GPU architectures by considering implicit deformable models such as image lattices (e.g., a 2D curve is implicitly represented as the iso-value of a field encoded as a 2D image). Level-set approaches have become particularly popular in the GPU-segmentation community as significant speedups and interactive rendering have become available.^[4,5] Geodesic active contours, which are a combination of traditional active contours (snakes) and level-set evolution, were efficiently implemented in GPU by using the total variational formulation and used primarily to separate foreground and background structures in 2D images.^[6] These implementations were, however, designed for the evolution of a single level set.

In this paper, we present a novel, computationally efficient framework for segmentation of histologic structures (e.g., nuclei and glands) in very large histopathology images. The framework presented alleviates the computational overhead of level set segmentation by marrying a very fast solver with an intuitive speed function. We leverage a powerful graphical processing unit (GPU) (via commodity graphics hardware) to provide a computationally optimized segmentation framework that allows for simultaneous calculation of multiple, interacting level sets. We demonstrate an application of our parallelized framework using our previously published synergistic active contour models (using multiple level sets)^[3,7] for the problem of segmenting nuclear and glandular structures on digitized histopathology. We have previously shown that this scheme^[3,7] allows for accurately detecting and segmenting overlapping lymphocytes and nuclei in H and E-stained prostate and breast needle core biopsy images. Following are the main contributions of this work:

- A GPU accelerated multiple level set-based hybrid active contour framework, implemented using the NVIDIA CUDA libraries, for the task of rapid segmentation of nuclei in H and E images.
- Leveraging a parallelizable framework to accommodate multiple level sets operating in parallel with initialization via shape prior in conjunction with

boundary and region-based energy terms.

- Exploit parallelism and efficient memory management in the GPU to achieve massive speedup compared to using a CPU.

The rest of the paper is structured as follows. Overview of GPU structure and programming paradigms are presented in section 2. Section 3 describes our segmentation algorithm. In Section 4 we describe our GPU-based framework and its implementation on graphics hardware. We present our analysis of the results in section 5. In section 6 we present our concluding remarks and discuss future improvements of our GPU-based implementation.

OVERVIEW OF GPGPU

General purpose computation on graphics processing units (GPGPU) is the technique of using graphics hardware to compute applications typically handled by the central processing unit (CPU). Graphics cards over the past two decades have been required to become highly efficient at rendering increasingly complex 3D scenes at high frame rates. This has forced their architecture to be massively parallel in order to compute graphics faster compared to general purpose CPUs.^[4] GPUs are not optimized for general purpose programs, and thus lack the complex instruction sets and branch control of the modern CPU. Although current high-performance CPUs feature multiple cores for limited parallel processing, GPUs are arguably a more attractive option in terms of lower price and power usage.

Recently, languages have been developed that allow the programmer to implement algorithms without any knowledge of graphics application programming interface (APIs) or architectures. The GPGPU computations performed in this work utilize the NVIDIA CUDA (Compute Unified Device Architecture) technology.^[8] The C language model has at its core three key abstractions:^[9] a hierarchy of thread groups (to allow for transparent scalability), shared memories (allowing access to low-latency cached memory), and barrier synchronization (to prevent *race conditions*). This breaks the task of parallelization into three subproblems, which allows for language expressivity when threads cooperate, and scalability when extended to multiple processor cores. CUDA allows a programmer to write *kernels* that when invoked execute thousands of lightweight identical threads in parallel. CUDA arranges these threads into a hierarchy of blocks and grids, as can be seen in Figure 1, allowing for runtime transparent scaling of code within GPU. The threads are identified by their location within the grid and block, making CUDA perfectly suited for tasks such as level set-based image segmentations where each thread is easily assigned to an individual *pixel* or *voxel*.^[10]

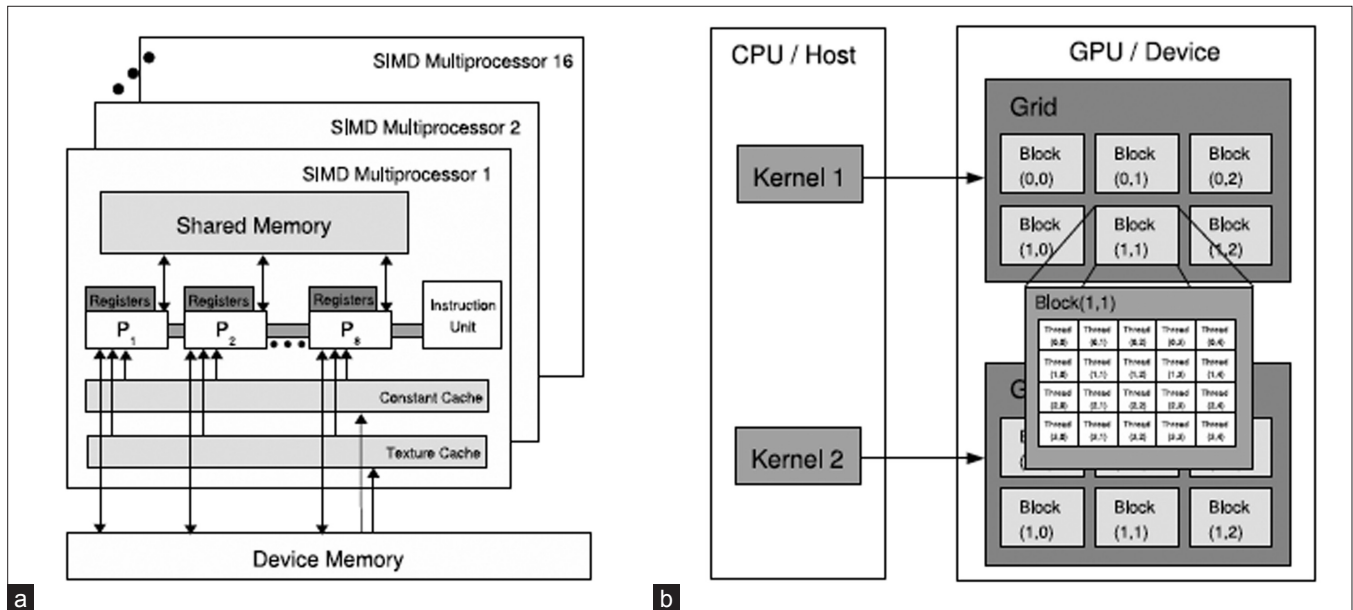


Figure 1: GPU architecture; (a) CUDA hardware interface, (b) CUDA software interface

NUMERICAL MODELING OF THE SEGMENTATION SCHEME

We present a slightly modified version of the previously published method by Ali *et al.*^[7] for the problem of segmenting nuclei and lymphocytes in prostate and breast cancer images. Our previous model comprised a hybrid active contour (boundary and region based) integrated with the shape prior. In this work, we eliminate the boundary-based term in order to simplify the numerical modeling of the variational formulation of the hybrid active contour formulation, relying instead on the region and shape prior terms within a multiple level set formulation. Figure 2 illustrates the individual modules comprising the scheme.

Integrated Shape-based Active Contour

We combine shape force and region force (Chan-Vese)^[11] into a variational formulation:

$$F = \underbrace{\beta_s \int_{\Omega} (\phi(x) - \psi(x))^2 dx}_{\text{Shape force}} + \underbrace{\beta_r \int_{\Omega} \theta_{in} H_{\phi} dx + \int_{\Omega} \theta_{out} H_{-\phi} dx}_{\text{Region force}} \quad (1)$$

where $\beta_s, \beta_r > 0$ are constants that balance contributions of the shape prior and the region term, $\{\phi\}$ is a level set function, and ψ is the shape prior. This formulation effectively integrates the shape prior with regional intensity information.

The level set formulation in Equation (1) is limited in that it allows for segmentation of only a single object at a time. To allow for simultaneous segmentation of multiple objects, we allow each pixel to be associated with multiple objects or the background instead of partitioning the image domain into mutually exclusive regions.^[7] Specifically, we try to find a set of characteristic functions

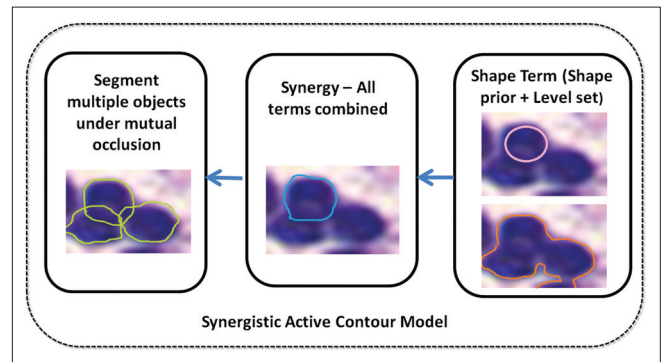


Figure 2: Flow chart showing the various modules comprising our segmentation scheme. First panel illustrates shape prior and region-based term. Second panel illustrates the synergy of both the terms

x_i such that we associate one level set per object in such a way that objects are allowed to overlap with each other within the image. Then simultaneous segmentation of two objects O_a, O_b with respect to ψ is solved by minimizing the following modified version of Equation (1):

$$F(\Phi, \Psi, u_{in}, u_{out}) = \beta_r \int_{\Omega} \Theta_{in} H_{x1vx2} dx + \int_{\Omega} \Theta_{out} - H_{x1vx2} dx + \omega \int_{\Omega} H_{x1vx2} dx + \sum_{i=1}^{N=2} \int_{\Omega} (\phi_i - \psi_i)^2 dx \quad (2)$$

with $H_{x1vx2} = H_{\psi1} + H_{\psi2} - H_{\psi1}H_{\psi2}, H_{x1vx2} = H_{\psi1}H_{\psi2}$, where $\Phi = (\phi_1, \phi_2)$ and $\Psi = (\psi_1, \psi_2)$. The fourth term penalizes the overlapping area between the two segmenting regions, and prevents the two evolving level set functions from becoming identical.

Discretized Iterative Scheme

Minimizing Equation (2) iteratively with respect to dynamic variables yields the associated Euler-Lagrange equations, parameterizing the descent direction by an

artificial time $t > 0$ as follows:

$$\begin{aligned} \frac{\partial \phi_i}{\partial t} &= \delta \phi_i \left[\mu \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right) - (((f - u_{in})^2 + (f - u_{out})^2)(1 - H_{\phi_i})) \right] - 2\nu(\phi_i - \psi_i) \\ \frac{\partial \phi_i}{\partial t} &= 2\nu \int_{\Omega} (\phi_i - \psi_i)(\nabla \psi_i \cdot \nabla_{\Theta} A_i) dx \\ \frac{\partial T_i}{\partial t} &= 2\nu \int_{\Omega} (\phi_i - \psi_i)(\nabla \psi_i \cdot \nabla_{\Gamma} A_i) dx \\ \frac{\partial s_i}{\partial t} &= 2\nu \int_{\Omega} (\phi_i - \psi_i) \left(-\frac{\psi_i}{s} + \nabla \psi_i \cdot \nabla_{s} A_i \right) dx \\ i, j &\in \{1, 2\}, i \neq j \end{aligned} \tag{3}$$

where ψ_i is defined in Ref.^[3]. Similar to the Chan-Vese model, we update u_{in} and u_{out} for each iteration as follows:

$$u_{in} = \frac{\int_{\Omega} f H_{x_1 \vee x_2} dx}{\int_{\Omega} H_{x_1 \vee x_2} dx}, u_{out} = \frac{\int_{\Omega} f(1 - H_{x_1 \vee x_2}) dx}{\int_{\Omega} (1 - H_{x_1 \vee x_2}) dx} . \tag{4}$$

The above model can be adapted for N objects (proof shown in Ref.^[3]). The solution to the equations in Equation (4) first requires discretization on a regular two-dimensional M -by- N grid. We let h denote the spacing between the cells and $(x_l, y_l) = (i_h, j_h)$ be the grid points with $0 \leq i \leq M$ and $0 \leq j \leq N$. Discretizing u_{in} and u_{out} is straightforward, as long as we fix a regularized Heaviside function. For our discretization we have chosen

$$H(z) = \begin{cases} 1, & \text{if } z > \epsilon \\ 0 & \text{if } z < -\epsilon \\ \frac{1}{2} + \frac{z}{2\epsilon} + \frac{1}{2\pi} \sin \left\{ \frac{\pi z}{\epsilon} \right\} & \text{if } |z| \leq \epsilon. \end{cases} \tag{5}$$

To discretize $\nabla \cdot \frac{\nabla_{\Theta} \phi}{|\nabla_{\Theta} \phi|}$ (i.e., divergence of the normalized gradient of ϕ), we let ϕ_x, ϕ_y denote partial derivatives of ϕ and similarly let ϕ_{xx}, ϕ_{yy} and ϕ_{xy} denote the second-order partial derivatives. Thus, we obtain

$$\nabla \cdot \frac{\nabla_{\Theta} \phi}{|\nabla_{\Theta} \phi|} = \frac{\phi_{xx} \phi_y^2 - 2\phi_{xy} \phi_x \phi_y + \phi_{yy} \phi_x^2}{(\phi_x + \phi_y)^{3/2}} \tag{6}$$

which we discretize with simple finite central differences:

$$\begin{aligned} \phi_x(i, j) &= \frac{(x_{i-1}, x_j) - (x_{i+1}, x_j)}{2h} \\ \phi_y(i, j) &= \frac{(x_i, x_{j-1}) - (x_i, x_{j+1})}{2h} \\ \phi_{xx}(i, j) &= \frac{(x_{i-1}, x_j) - (x_{i+1}, x_j) + 2(x_i, x_j)}{h^2} \\ \phi_{yy}(i, j) &= \frac{(x_i, x_{j-1}) - (x_i, x_{j+1}) / 2 + 2(x_i, x_j)}{h^2} \\ \phi_{xy}(i, j) &= \frac{1}{4h^2} ((x_{i+1}, x_{j+1}) + (x_{i-1}, x_{j+1}) + (x_{i+1}, x_{j-1}) + (x_{i-1}, x_{j-1})). \end{aligned} \tag{7}$$

Algorithm 1: Sequential level set segmentation

Input: Feature image I , Initial Mask m , threshold T , Range ϵ
Iterations n , Reinitialize Every RI

Output: Nuclear contours (boundaries)

Initilize ϕ_0 to Signed Euclidean Distance Transform (SEDT) from mask m

Calculate Data Speed Term $D(I) = \epsilon - |I - T|$ For all the n Iterations do

 Calculate First Order Derivatives ϕ_x, ϕ_y

 Calculate Second Order Derivatives $\phi_{xx}, \phi_{yy}, \phi_x \cdot \phi_{xy}$

 Calculate Curvature Terms n^+, n^-

 Calculate Gradient $\nabla \phi$

 Calculate Speed Term $F = \alpha D(x) + (1 - \alpha) \nabla \cdot \frac{\nabla_{\Theta} \phi}{|\nabla_{\Theta} \phi|}$

 Update Level Set Function $\phi(t + \Delta t) = (t) + \Delta t F |\nabla \phi|$

 if Iterations % $RI = 0$ then

 | Reinitialize ϕ to SEDT

 end

end

Sequential Algorithm and Implementation

The iterative evolution of the level set ϕ is described in algorithm 1 using the individual components defined in section 3.2. We start by initializing ϕ with some initial contour (an ellipse) and set $n=0$. We then compute u_{in} (ϕ^n) and u_{out} (ϕ^n) from Equation (4). For computing curvature forces, n^+, n^- , we direct the reader to the formulation.^[11] Next we compute Equation (3) with the given discretization above to obtain $\phi^{n+1} = \phi^n + \Delta t \frac{\partial \phi}{\partial t}$ with explicit forward Euler. Finally, we update $n = n + 1$. Note that we initialize the distance function every RI^{th} (user defined parameter) iteration.

GPU IMPLEMENTATION OF INTEGRATED SHAPE-BASED ACTIVE CONTOUR SCHEME

The GPU-based implementation of the segmentation scheme (described in section 3) is performed with the NVIDIA CUDA toolkit.

Parallel Algorithm and Implementation

The parallel implementation of the segmentation scheme follows the structure shown in algorithm 2. Input image, I , and an initial contour ϕ (an ellipse) are both discretized and generated on equally sized 2D grids on CPU. We copy both the image and ϕ onto the GPU. From this point on, the GPU has everything it needs and requires no further copying of data, thus effectively optimizing memory transfers between the CPU and GPU.

Kernel Threads Setup

In CUDA, it is assumed that both the host (CPU) and device maintain their own DRAM.^[8] Host memory is allocated using malloc and device memory is allocated using cudaMalloc. Since memory bandwidth between the host memory and device memory is low (it is much lower than the bandwidth between the device and the device

Algorithm 2: Parallel Implementation

```

Initialize  $\phi_{ij}^0, D$  on host memory
Allocate memory for  $\phi^n, \phi^{n+1}, D$  on device
Copy  $\phi^0, D$  from host device for all the  $n$  iterations do
    Execute Level Set Update CUDA Kernel  $\phi_{ij}^{n+1} = \phi_{ij}^n + \Delta t F |\nabla \phi_{ij}^n|$ 
    Swap pointers of  $\phi_{ij}^n, \phi_{ij}^{n+1}$ 
    if Iterations % RITS == 0 then
        Copy  $\phi$  from device to host
        Reinitialize  $\phi$  to Signed Euclidean Distance Transform
        Copy  $\phi$  from host to device
    end
end
Copy  $\phi$  from device to host

```

memory), it is recommended to keep the number of transfers to a minimum. In order to minimize the latency of accessing the shared memory it is recommended to make the block size a multiple of 16 and use the cudaMallocPitch routine to allocate memory with padding if the X dimension of the image is not a multiple of 16. Hence, most CUDA programs follow a standard structure of (1) initialization, (2) host to device data transfer, (3) performing computations, and (4) finally memory transfer of computed results from device to host.

CUDA threads are assigned a unique thread ID that identifies its location within the thread block and grid. This provides a natural way to invoke computation across the image and level set domain, by using the thread IDs for addressing. This is best explained with Table 1. Assume that our image has dimensions 4×4 and the block size is 2×2 . Invoking the kernel with a grid size of 2×2 blocks results in the 16 threads shown in Table 1, in the form (threadIdx.y, threadIdx.x). These threads are grouped into blocks of four.

As each thread has access to its own threadIdx and blockIdx, global indices (i, j) can be determined using the equations,

$$\begin{aligned} \text{int } i &= \text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x}; \\ \text{int } j &= \text{blockIdx.y} * \text{blockDim.y} + \text{threadIdx.y}; \end{aligned} \quad (8)$$

where blockDim.x and blockDim.y represent the dimensions of the block (which in this case are both equal to 2). Of course, much larger block sizes are used, keeping the block X dimension (BX) a multiple of 16 for maximum speed. The effect of different block sizes on performance is analyzed in section 5.5. Once these indices are set up, it is relatively straightforward to transfer the level set update code to a CUDA kernel.

2D Shared Memory Optimization

Since we use finite differences to compute the curvature force in a grid cell, we need to access the value of ϕ from the neighboring cells (blocks). Unfortunately the GPU is not optimized for this sort of memory access

Table 1: Sample arrangement of 16 threads grouped into blocks of 4

(0,0)	(0,1)	(0,1)	(0,1)
(1,0)	(1,1)	(1,0)	(1,1)
(0,0)	(0,1)	(0,0)	(0,1)
(1,0)	(1,1)	(1,0)	(1,1)

pattern, and memory accesses can suffer a significant performance penalty.^[8] In order to keep the number of costly accesses to device memory at a minimum, effective use of the on-chip shared memory is essential. This along with maximizing parallel execution and optimization of instruction usage form the three main performance optimization strategies for CUDA.^[8]

Integrating use of the shared memory into the CUDA kernel requires partitioning the level set domain into tiles. For first-order finite difference problems such as ours, each tile must also contain values for neighbor nodes (often known as *halo* nodes) for $i+1$ and $j+1$ elements, which would be stored in separate tiles, so these must also be read into shared memory. As the size of the shared memory is only 16 KB, the sizes of the tiles and corresponding halo are limited. Micikevicius^[12] outlined a framework for handling such a process. While such a process may serve as a good model for a multi-GPU implementation, the kernel will need to be modified as it is optimized for higher order stencils (without cross-derivative terms). Instead, tiling code was adapted from Giles' Jacobi iteration for Laplace discretization algorithm^[13] which supports cross-derivatives well. The shared memory management technique in this finite difference algorithm accelerated the global memory implementation by over an order of magnitude.

For a block (and tile) size of $BX \times BY$ there are $2 \times (BX + BY + 2)$ halo elements, as can be seen in Figure 3 where darker elements represent the thread block (the active tile) and the lighter elements represent the halo. It is in this manner that the domain of the computation is partitioned and results in overlapping of halo nodes.

Each thread loads ϕ_n values from global memory to the active tile stored in shared memory. However, depending on the location of the thread within the thread block it may also load a single halo node into the shared memory. Therefore in order to load all halo nodes, this technique assumes that there are at least as many interior nodes as there are halo nodes. Before data can be loaded into the halos, the thread ID needs to be mapped to the location of a halo node both within the halo and within the global indices.

The first $2 \times (BX + BY + 2)$ threads are assigned to load values into the halo in this manner. This is best visualized with the example of a 6×6 thread block as shown in Figure 3b. This method of loading elements has been chosen in order to maximize *coalescence*, the merging

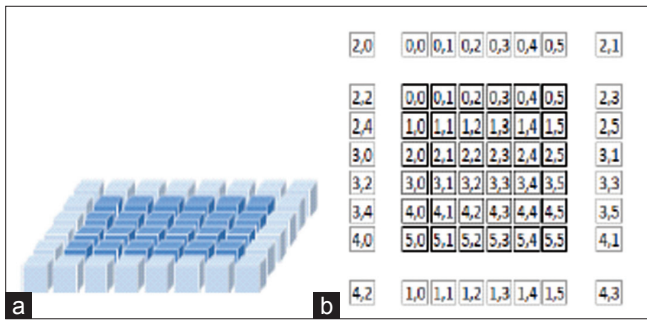


Figure 3: (a) 2D shared memory arrangement; (b) tile and halo showing a block mapping of thread IDs to halo nodes

of adjacent blocks of memory to fill gaps caused by deallocated memory. Not only are the interior tile nodes loaded coalesced, but as can be seen above, the first 12 elements of the thread block load the y halos (above and below the interior tile excluding corners) in a coalesced manner. The side halos (x halos) loads are noncoalesced. When writing back results to global memory, as only the interior nodes have updated values they are written to global memory coalesced.

Calculation of Forces

To compute u_{in} and u_{out} , we start by computing the value of u_{in} and u_{out} in each grid cell, storing it in a temporary array. Since these are scalar values, we use reduction operation,¹ cutting the time spent to $O(\log n)$ time assuming we have an infinite number of threads. In a reduction, a binary function is performed on all elements in an array; this can be done in a tree-like fashion, where the given operation can be performed concurrently on different parts of the array and allows for combining the results at the end. NVIDIA has created an STL-like C++ library called thrust that, among other things, can do reductions efficiently on the GPU.

Hereafter, we use the values for u_1 and u_2 computed above to compute the image forces. By image forces, we mean the terms $\lambda_1(f-u_1)^2 + \lambda_2(f-u_2)^2$ and the F_{shape} from Equations (3) and (4). This is straightforward and coalesced (reader is referred to CUDA guide).^[9] Lastly, we need to update ϕ for the next iteration, which is straightforward since we have the two grids containing the curvature force and the image forces. The final step is to copy the new ϕ back to the CPU.

EXPERIMENTAL RESULTS AND DISCUSSION

Model Parameters, Implementation, and Initialization

In this paper, for the shape model, we generate a training set of 30 ellipses (nuclei and lymphocytes being

¹Reduction operation: Variable has a local copy in each thread, but the values of the local copies will be summarized (reduced) into a global shared variable.

elliptical in shape) by changing the size of a principal axis with a Gaussian probability function. The manner of choosing the weighting parameters from Equations (1) and (4) is as follows: $\lambda_1=1$, $\lambda_2=2$, and $\mu=0.2$ determine the size of the neighborhood where the gray value information is averaged by means of diffusion, β_r and β_s are chosen such that the shape prior is attracted toward the region to be segmented. Level sets are manually initialized and the model assumes that the level sets are strategically placed in the vicinity of the objects. Some features such as the Courant–Friedrichs–Lewy (CFL) condition could not be implemented in this parallel version without slowing down computation time significantly.^[6] This is because such a condition requires the determination of the largest element of $\nabla\phi$ which is computed roughly half way through the update procedure. Therefore integrating this condition would require transferring $\nabla\phi$ and curvature terms back to host memory to determine $\max\{F|\nabla\phi|\}$, or perhaps more efficiently calling a CUDA kernel to determine the largest element. The cost of this added complexity and slowdown outweighed the benefits, and therefore ∇t was chosen to be a fixed parameter.

All hardware testing was done on a single PC with an Intel Core 2 Duo T8100 Processor with a clock speed of 2.1 GHz and 4 GB of RAM. The graphics hardware used was the NVIDIA GeForce 8600M GT, with CUDA 2.2 software installed. Timing code used was from the cutil library provided in the CUDA toolkit.

Data Description

Evaluation is done on two different histopathology datasets: prostate cancer and breast cancer cohorts comprising 14 and 52 images respectively [Table 2]. A total of 70 cancer nuclei from 14 images for prostate and 504 lymphocytes from 52 images for breast cancer were manually delineated by an expert pathologist (serving as the ground truth annotation for quantitative evaluation). For both datasets the objective was to detect, segment the individual nuclei and lymphocytes, and where appropriate, resolve the overlap between intersecting objects. To evaluate the speedup, various patch sizes were created from each of the data set [Table 3].

Evaluation of Segmentation and Detection Accuracy

Results shown in Figure 4 for nuclei segmentation aim to demonstrate the strength of our model in terms of detection, segmentation, and overlap resolution accuracy. Table 4 shows the segmentation accuracy (individual lymphocytes and nuclei) compared to those of Chan-Vese^[11] (M^{CV}) and Ali *et al.*^[3] (M^{Ali}). Note that GPU-based implementation (M^{GPU}) underperforms in comparison to M^{Ali} since M^{GPU} lacks the hybrid boundary based term present in M^{Ali} . Figure 5 showcases qualitative results of nuclear segmentation achieved by the three models.

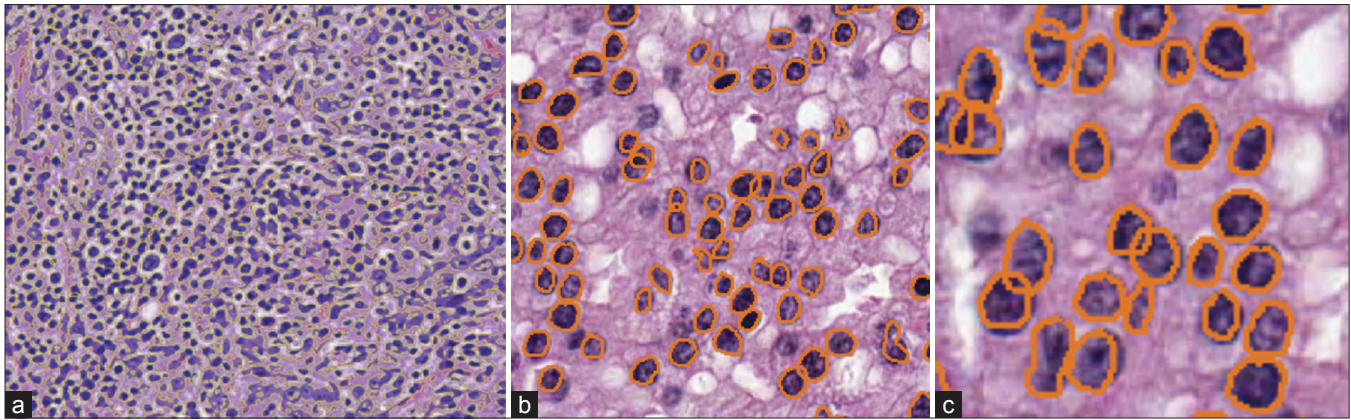


Figure 4: Segmentation results from our model applied to histological image patches of sizes (a), (b) 256x256, (c) 128 x 128 pixels. Note that our model is able to segment intersecting, overlapping nuclei

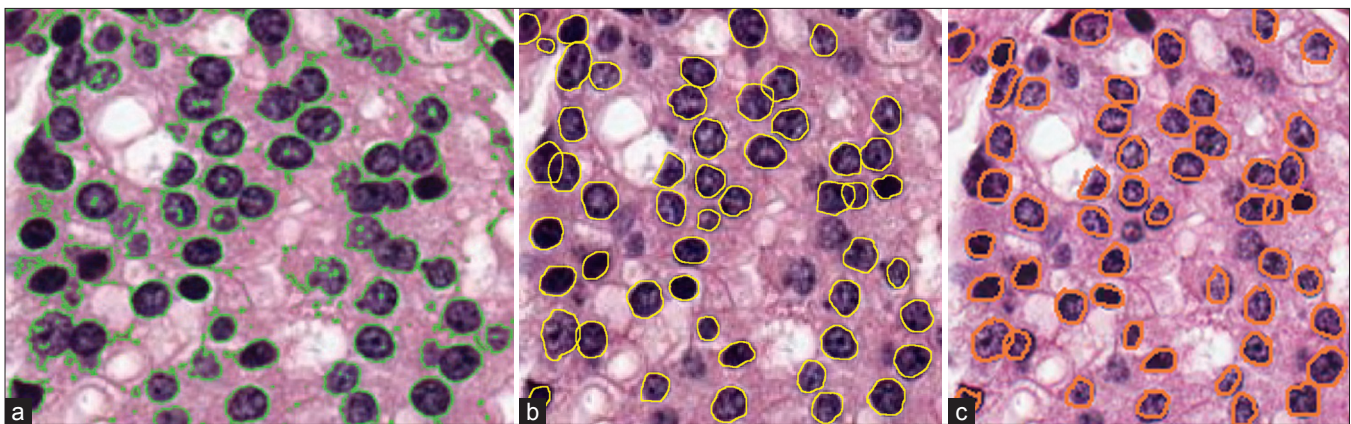


Figure 5: Segmentation results comparing (a) traditional Chan-Vese model^[11], (b) Ali et al.,^[13] our GPU-accelerated model

Table 2: Description of the different data sets considered in this study

	Name	Number
Dataset 1	Hematoxylin and Eosin (H and E) stained prostate needle core biopsy images	14
Dataset 2	Hematoxylin and Eosin (H and E) stained breast biopsy images	52

Table 3: Description of patch sizes from two data sets for which M^{GPU} and speedup was evaluated

	Dataset 1	Dataset 2
Patch size	No. of images	No. of images
512 × 512	4	15
256 × 256	5	17
128 × 128	5	20

We compute SN, PPV, OR (overlap detection ratio) for each image, and then determine the average and standard deviation across the 62 images. Note that overlap detection ratio $OR = \frac{\# \text{ overlap resolved}}{\text{Total} \# \text{ of overlaps}}$ is defined as the fraction of total overlaps successfully resolved by the segmentation scheme. These statistics are reported in Table 4 and they

reflect the efficacy of M^{GPU} in segmenting nuclei and lymphocytes in CaP and BC images.

Evaluation of Speedup

We present results of various speedups achieved on data set containing image sizes 256×256 and 512×512 . First we compare the average time taken for 2000 iterations in MATLAB, C, and CUDA on histopathology data of good contrast and dimensions 256×256 (a multiple of 16 implying no memory padding is required in CUDA). The results shown in Table 5 reflect the speedup achieved by our GPU-accelerated method.

The average runtime speedup attained from sequential code in C to CUDA-optimized code is approximately 13×. The block size used for 2D CUDA compute was 32×8 . In another experiment, we selected a set of images with relatively poor contrast and dimensions 512×512 . This makes the image both a computationally demanding segmentation (as it has relatively large dimensions) and challenging in terms of accuracy. The average performance speedup attained on this larger image set was 19×, greater than the speedup attained for the smaller 256×256 images. This motivates exploration into the effect of different image sizes on CUDA speedup, discussed in the next section.

Table 4: Quantitative evaluation of segmentation results between models over 66 images

	SN	PPV	OR
M ^{CV}	0.23	0.57	0.09
M ^{Ali}	0.82	0.67	0.91
M ^{GPU}	0.73	0.64	0.87

Table 5: Comparison of average runtimes for our segmentation algorithm in different programming environments. All comparisons were performed for image patches of size 256 × 256 and 512 × 512

	256 × 256	512 × 512
Algorithm Version	Time (s)	Time (s)
MATLAB	2615.95	4415
C	960.44	1320.5
CUDA (M ^{GPU})	80.73	69.4

Evaluation of the Effects of Image Size on Speedup

Figure 6 shows the effect of multiple patch sizes on computational time for up to 1000 iterations on the optimized CUDA algorithm. It can be seen from Figure 6 that the speedup for smaller sized image patches is relatively less compared to the speedup obtained for images with larger sizes. The sequential algorithm performs almost half as slowly for volume sizes larger than 64². This is most likely due to the fast on board CPU cache being used for smaller volume sizes (<64²) volume sizes larger cannot fit on the CPU cache and hence are stored on the slower DRAM. Conversely, the CUDA code performs relatively poorly for smaller image sizes and much more quickly for larger images. This is essentially due to low numbers of processors being used for such small images and many more being used for larger images. Therefore the speedup line essentially shows that the algorithm follows Amdahl's and Gustafson's laws of parallel computation.^[4]

Image sizes much larger than 256² could not be tested as the maximum amount of global memory available for the 8600M GT is 256 MB. For example, an 320² image would take up 320² × size of (float) and there are three of these arrays (for the image, previous level set iteration and current level set iteration), which would take up 295 MB of graphics memory. It is however expected, for these even larger volumes, that the speedup of the algorithm will remain at the observed plateau [Figure 6].

CONCLUSIONS

We presented a novel GPU accelerated segmentation scheme that can detect and segment multiple overlapping objects in an image. Our implementation was able to significantly reduce the computational time required for the popular segmentation models, including

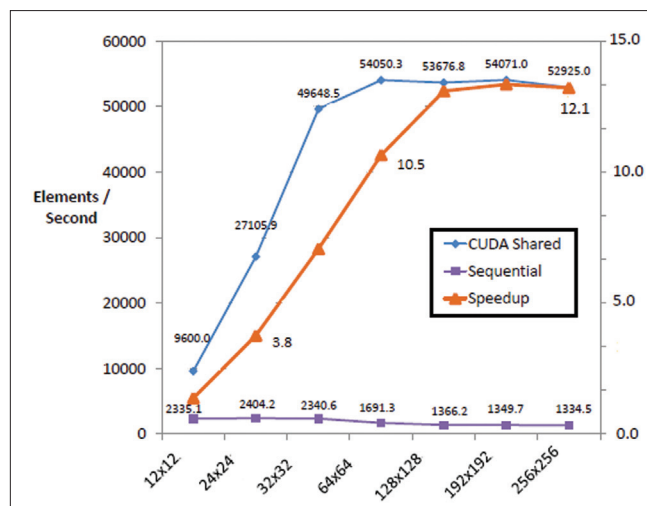


Figure 6: Number of elements computed per second for different volume sizes

an integrated active contour scheme with shape priors, inspired by Ali *et al.*^[3] In future work, we intend to extend the framework to incorporate automated model initialization and to evaluate the model on larger image sizes. To further improve the segmentation results, we plan to incorporate a hybrid boundary and region based model with Courant-Friedrichs-Lewy (CFL) condition without compromising the speedup.

Computationally inexpensive segmentation schemes that can accurately segment structures on digitized histology images, especially the ability to deal with overlapping and occluded structures, are highly critical in the context of a number of different diagnostic and prognostic applications. We applied our parallelized algorithm for segmentation and detection of lymphocytes and nuclei in breast and prostate histopathology imagery. We evaluated our algorithm against two popular platforms and with varying image sizes. Our results show that our GPU-accelerated framework was able to segment nuclei and lymphocytes with a speedup of 19×.

ACKNOWLEDGMENTS

Thanks to funding agencies: National Cancer Institute (R01CA136535-01, R01CA140772 01, R03CA143991-01), and The Cancer Institute of New Jersey. We would also like to thank Drs. Michael Feldman, John Tomaszewski, Natalie Shih, and Shridar Ganesan for providing the digitized data used in this study.

REFERENCES

1. Madabhushi A. Digital pathology image analysis: Opportunities and challenges (editorial). *Imag Med* 2009;1:7-10.
2. Basavanahally A, Ganesan S, Agner S, Monaco J, Feldman M, Tomaszewski J, et al. "Computerized Image-Based Detection and Grading of Lymphocytic Infiltration in HER2+ Breast Cancer Histopathology," *IEEE Trans Biomed Eng* 2010;57:642-53.
3. Ali S, Madabhushi A. "Active Contour For Overlap Resolution Using

- Watershed Based Initialization (ACOReW):Applications To Histopathology". IEEE Int Symp Biomed Imag: From Nano to Macro 2011. p. 614-7.
4. Kilgariff E, Fernando R. GPU Gems 2, ch. The GeForce 6 Series GPU Architecture. Boston: Addison Wesley; 2005. p. 471491.
 5. Ruiz A, Kong J, Ujaldon M, Boyer KL, Saltz JH, Gurcan MN. "Pathological image segmentation for neuroblastoma using the GPU". Biomedical Imaging: From Nano to Macro, IEEE International Symposium on, 2008. p. 296-9.
 6. Lefohn AE, Whitaker R. "GPU based, three-dimensional level set solver with curvature flow", technical report uucs-02-017, School of Computing, University of Utah, 2002.
 7. Ali S, Madabhushi A. "Segmenting multiple overlapping objects via an integrated region and boundary based active contour incorporating shape priors: Applications to histopathology", SPIE Med Imag 79622W (2011).
 8. NVIDIA CUDA Programming Guide ver 2.2.1. Available from: <http://www.nvidia.com/object/cuda-develop.html>. [Accessed May 2011].
 9. Schmid J, Guitin I, Jos, Gobbetti E, Magnenat-Thalmann N. "A GPU framework for parallel segmentation of volumetric images using discrete deformable model". Vis Comput 2011;27:85-95.
 10. Tejada E, Ertl T. "Large steps in GPU-based deformable bodies simulation. Simulat Model Pract Theory 2005;13:703715.
 11. Chan TF, Vese LA. "Active contours without edges", IEEE Trans. on Image Process 2001;10:266-77.
 12. Micikevicius P. "3D Finite difference computation on GPUs using CUDA", In GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graph Processing Units, ACM. New York, NY, USA, (2009. p. 79-84.
 13. Giles M. Jacobi iteration for a laplace discretisation on a 3rd structured grid 2008. Available at people.maths.ox.ac.uk/gilesm/cuda/prac3/laplace3d.pdf.