Published in final edited form as:

Proc IEEE Int Conf Healthc Inform Imaging Syst Biol. 2011 July ; 2011: 174–181. doi:10.1109/HISB.2011.10.

Acceleration of Two Point Correlation Function Calculation for Pathology Image Segmentation

Lee A.D. Cooper,
Center for Comprehensive Informatics, Emory University, Atlanta, GA

Joel H. Saltz,
Center for Comprehensive Informatics, Emory University, Atlanta, GA

Umit Catalyurek, and
Department of Biomedical Informatics, The Ohio State University, Columbus, OH

Kun Huang
Department of Biomedical Informatics, The Ohio State University, Columbus, OH

Abstract

The segmentation of tissue regions in high-resolution microscopy is a challenging problem due to both the size and appearance of digitized pathology sections. The two point correlation function (TPCF) has proved to be an effective feature to address the textural appearance of tissues. However the calculation of the TPCF functions is computationally burdensome and often intractable in the gigapixel images produced by slide scanning devices for pathology application. In this paper we present several approaches for accelerating deterministic calculation of point correlation functions using theory to reduce computation, parallelization on distributed systems, and parallelization on graphics processors. Previously we show that the correlation updating method of calculation offers an 8–35× speedup over frequency domain methods and decouples efficient computation from the select scales of Fourier methods. In this paper, using distributed computation on 64 compute nodes provides a further 42× speedup. Finally, parallelization on graphics processors (GPU) results in an additional 11–16× speedup using an implementation capable of running on a single desktop machine.

Keywords

Microscopy; Image Segmentation; Digital Pathology; Two Point Correlation Function; Graphical Processing Unit

I. Introduction

With the development of high resolution digital scanning imaging technology, huge amount of histology images are being generated in both clinical applications and basic research projects. In both situations, these images need to be analyzed by personnel with pathology training for identifying cellular and tissue structures of interest. However, the large image size and quantity are prohibiting from accurate quantitative analysis of such images in many cases. For instance, the histology image for a specimen of size 1 cm-by-1 cm, when scanned

with 40X objective lens, will yield an image of about 40, 000 × 40, 000 pixels. Thus such large images require the development of (semi)automatic computer based methods for segmenting specific objects of interest [1], [2].

In our previous work, we have demonstrated that the two-point correlation function (TPCF) is an effective image feature for segmenting different types of tissues in digitized histology images [3], [4], [1], [5]. However they are accompanied by a significant computational burden. Consider the following example: computing TPCF features for a $16K \times 16K$-pixel image with four components implies the calculation of more than one billion 2D correlations. Performing these correlations is a considerable task, with large image datasets pushing the correlation calculations into the trillions. Recently, using a *correlation updating* technique, we were able to significantly improve the computational efficiency by up to 67 times. This technique uses a derived relationship between TPCFs of neighboring regions-of-interest to update TPCF values rather than computing them from scratch. Thus we do not waste computation on unused correlation values, and that eliminates the strong time-dependency on window size that exists for FFT-based correlation.

However, since the computation of TPCF as a local image feature for image segmentation is a local operation, it is highly parallelizable. Thus it is expected that parallel computing will further reduce the running time. In this paper, we take two approaches. The first approach is the parallelization of TPCF feature calculations on the multi-node and multi-socket levels. The second approach is the implementation of correlation updating using graphics processors (GPU), taking advantage of the fine-grained parallelism and fast on-chip memory to further optimize TPCF feature calculation.

II. Two Point Correlation Function

We have given detailed description of local TPCF features for an image and how to use them for image segmentation in our previous work [3], [4], [1], [5]. However, for completeness, we reiterate the formulation of TPCF, its calculation and the correlation updating process in this section.

The computation of TPCF features is depicted in Figure 1. The computation consists of three main processes: correlation calculation, normalization, and sampling/interpolation.

Given an $M \times N$ digital phase image I, a $w \times w$ region-of-interest (ROI) $\Phi_{x,y}$ is defined with upper left corner $I(x, y)$. For each phase i in the ROI, the autocorrelation of the binary mask $\mathcal{I}^{(i)}_{x,y}$ is calculated

$$R^{(i)}(\Delta x, \Delta y) = \sum_m \sum_n \mathcal{I}^{(i)}_{x,y}(m,n)\mathcal{I}^{(i)}_{x,y}(m+\Delta x, n+\Delta y), \quad (1)$$

where $\Delta x, \Delta y \in \mathbb{Z}$. The values of R_i are normalized by the number of overlapping pixels to calculate probabilities

$$\hat{R}^{(i)} = R^{(i)} ./ (\mathbf{1}_{M \times N} * \mathbf{1}_{M \times N}), \quad (2)$$

where $\mathbf{1}_{M \times N}$ is an $M \times N$ matrix of ones, ./ is element-wise division, and * is convolution.

The normalized elements of $\hat{R}$ represent the homogeneous anisotropic TPCF $S_2^{(i)}(\mathbf{x})$. The isotropic quantity $S_2^{(i)}(r)$ is calculated using the process of circumferential sampling depicted in Figure 2. Samples taken at a distance r from $\hat{R}^{(i)}(0, 0)$ are averaged over angle

$$S_2^{(i)}(r) = \frac{\Delta\theta}{\pi} \sum_{k=0}^{\frac{\pi}{\Delta\theta} - 1} \hat{R}^{(i)}(r \cos(k\Delta\theta), r \sin(k\Delta\theta)), \quad (3)$$

where $\Delta\theta$ is the *angular interval*. Samples that do not fall on the discrete grid of $\hat{R}^{(i)}$ can be inferred using bilinear interpolation. Due to the symmetry of $\hat{R}^{(i)}$, the sampling angles can be restricted to $[0, \pi)$.

This procedure is repeated for every phase i in the ROI $\Phi_{x,y}$ to calculate the feature vector $v_{x,y}$. The ROI is positioned at every complete location in the phase image $(x, y) \in \{0, 1, \ldots, N-w\} \times \{0, 1, \ldots, M-w\}$ to generate a set of $(M - w + 1)(N - w + 1)$ feature vectors.

III. Direct FFT-based correlation

The most computationally demanding portion of the TPCF calculations are the correlations of Equation 1. These correlations may be computed efficiently using the Fast Fourier Transform (FFT). The binary mask $\mathcal{I}_{x,y}^{(i)}$ is padded to the size $2w - 1$

$$\mathcal{P}_{x,y}^{(i)} \equiv \begin{bmatrix} \mathcal{I}_{x,y}^{(i)} & \mathbf{0}_{w \times w-1} \\ \mathbf{0}_{w-1 \times w} & \mathbf{0}_{w-1 \times w-1} \end{bmatrix} \quad (4)$$

and transformed forward to the discrete frequency domain

$$\mathcal{F}[k, l] = \frac{1}{\sqrt{(2w-1)}} \sum_{n=0}^{2w-1} \sum_{m=0}^{2w-1} \mathcal{P}_{x,y}^{(i)}[m, n] e^{-2\pi j \frac{mk + nl}{2w-1}}. \quad (5)$$

The power spectrum is calculated by taking the magnitude of the complex elements $\mathcal{F}[k, l]$ and the inverse transformation is computed to obtain the autocorrelation R

$$R^{(i)} = \frac{1}{\sqrt{(2w-1)}} \sum_{l=0}^{nw-1} \sum_{k=0}^{2w-1} \mathcal{F}_{x,y}^{(i)}[k,l] e^{2\pi j \frac{mk+nl}{2w-1}}. \quad (6)$$

The dimension $2w-1$ is critical for the performance of the FFT calculations. The most widely used FFT library, FFTW [6], offers optimal performance for powers of two or small prime factors. The padding of Equation 4 may be manipulated to achieve these sizes, only by adding zeros to achieve the next most favorable size. A demonstration of the effects of transform size and padding is presented in Section VIII.

A. Sparse sampling

The FFT calculates all $(2w-1)^2$ elements of the autocorrelation R, however only a small set of these are required for the circumferential sampling procedure of Equation 3. This is apparent in Figure 2, where for $w = 32$ and $\Delta\theta = \pi/8$ only 10% elements of $R^{(i)}$ are used to interpolate $S_2^{(i)}(r)$. Although algorithms exist for computing subsets of FFT outputs [7], [8], [9], the available implementations of ordinary full-output FFT are optimized to the extent that only a relatively large transform will benefit [6].

B. Correlation updating

In addition to the sampling sparsity, the shared content between neighboring ROIs also points to significant amounts of wasted computation. For example, although $\Phi_{x,y}, \Phi_{x+1,y}$ differ by only two w-length columns of pixels, a straight-forward FFT method calculates correlations from scratch for each region.

The observations of sparsity and shared content may be simultaneously addressed using the linearity of correlation. Rather than computing $R^{(i)}$ from scratch for each ROI, the portions of neighboring ROIs, say $\Phi_{x,y}$ and $\Phi_{x+1,y}$, that are not shared may be used to update $R^{(i)}$ from $\Phi_{x,y}$ to $\Phi_{x+1,y}$ instead. Furthermore, if this updating is performed directly in the image domain then the locations used in sampling may be selectively updated, and the spatial dependency between the image and frequency domains can be avoided.

Given two horizontally adjacent $w \times w$ ROIs $\Phi_{x,y}, \Phi_{x+1,y}$ with corresponding indicators

$$\begin{aligned} \mathcal{I}_{x,y}^{(i)} &= [c_x, c_{x+1}, \ldots, c_{x+w-1}] \\ \mathcal{I}_{x+1,y}^{(i)} &= [c_{x+1}, c_{x+2}, \ldots, c_{x+w}], \end{aligned} \quad (7)$$

where c are w-length columns of pixels. The autocorrelation of $\mathcal{I}_{x,y}^{(i)}$ is denoted $R_{x,y}^{(i)}$. Given that $I_{x,y}^{(i)}, I_{x+1,y}^{(i)}$ are distinguished only by c_x, c_{x+w+1}, the autocorrelation $R_{x+1,y}^{(i)}$ can be calculated from $R_{x,y}^{(i)}$ by adding the contribution of c_{x+w+1} and removing the contribution of c_x.

Define the correlation sums between the columns and their respective regions

$$a^{-}_{\Delta x,\,\Delta y} \equiv \sum_{m} \mathcal{I}^{(i)}_{x,\,y}(\Delta x, m)\, c_x(m+\Delta y) \qquad (8)$$
$$a^{+}_{\Delta x,\,\Delta y} \equiv \sum_{m} \mathcal{I}^{(i)}_{x+1,\,y}(\Delta x, m)\, c_{x+w}(m+\Delta y)\,.$$

The *update matrices* containing these correlation sums represent the contributions of c_x to $R^{(i)}_{x,\,y}$ and c_{x+w+1} to $R^{(i)}_{x+1,\,y}$

$$A^{-} \equiv \begin{bmatrix} a^{-}_{w-1,\,w-1} & \cdots & a^{-}_{0,\,w-1} & a^{-}_{1,\,1-w} & \cdots & a^{-}_{w-1,\,1-w} \\ a^{-}_{w-1,\,w-2} & \cdots & a^{-}_{0,\,w-2} & a^{-}_{1,\,2-w} & \cdots & a^{-}_{w-1,\,2-w} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a^{-}_{w-1,\,0} & \cdots & a^{-}_{0,\,0} & a^{-}_{1,\,0} & \cdots & a^{-}_{w-1,\,0} \\ a^{-}_{w-1,\,-1} & \cdots & a^{-}_{0,\,-1} & a^{-}_{1,\,1} & \cdots & a^{-}_{w-1,\,1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a^{-}_{w-1,\,1-w} & \cdots & a^{-}_{0,\,1-w} & a^{-}_{1,\,w-1} & \cdots & a^{-}_{w-1,\,w-1} \end{bmatrix},$$

$$A^{+} \equiv \begin{bmatrix} a^{+}_{0,\,1-w} & \cdots & a^{+}_{w-1,\,1-w} & a^{+}_{w-2,\,w-1} & \cdots & a^{+}_{0,\,w-1} \\ a^{+}_{0,\,2-w} & \cdots & a^{+}_{w-1,\,2-w} & a^{+}_{w-2,\,w-2} & \cdots & a^{+}_{0,\,w-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a^{+}_{0,\,0} & \cdots & a^{+}_{w-1,\,0} & a^{+}_{w-2,\,0} & \cdots & a^{+}_{0,\,0} \\ a^{+}_{0,\,1} & \cdots & a^{+}_{w-1,\,1} & a^{+}_{w-2,\,-1} & \cdots & a^{+}_{0,\,-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a^{+}_{0,\,w-1} & \cdots & a^{+}_{w-1,\,w-1} & a^{+}_{w-2,\,1-w} & \cdots & a^{+}_{0,\,1-w}. \end{bmatrix}$$

The relationship between the autocorrelations for adjacent regions is then

$$R^{(i)}_{x+1,\,y} = R^{(i)}_{x,\,y} - A^{-} + A^{+}\,. \qquad (9)$$

This updating procedure clearly applies to vertically adjacent ROIs as well.

Since only a subset of the elements of R are required for sampling, the corresponding update elements of A^{+},A^{-} may be calculated individually for the sampling locations. Each sampling location will then require only $2w$ multiply-add operations for updating from one ROI to the next. Given the updating procedure, to calculate TPCF features over an entire phase image requires only P total FFTs to initialize $R^{(i)}_{0,\,0}$, $i = 1, \ldots, P$. With the initialization calculated the

updating procedure is used to iterate the ROI both horizontally and vertically through the remaining positions.

The updating procedure does not compromise numerical accuracy in the calculation of autocorrelation. Since the elements of R represent counts of pixels with a given separation there is no accumulation of error through repeated rounding operations. The updating procedure also provides flexibility in choosing the ROI size w since performance is not subject to the restrictions on FFT size.

Results comparing the time performance of updating against the ordinary FFT method are presented in Section VIII.

IV. Parallelization

The procedure for calculating TPCF feature vectors from a phase image is a simple data parallelism. The image may be divided among different nodes, sockets, or cores, with each resource computing TPCF features for a separate portion of the image.

The implementation used for the experiments of this chapter assumes a head/worker organization. A single node loads the phase image, partitions it into horizontal strips and distributes the strips to processing elements (including itself) using asynchronous communication and double buffering to overlap communication with disk operation. Each node calculates the TPCF features for its portion of the image and returns the results to the head node. Message Passing Interface (MPI) is used for communication between sockets and nodes [10] to achieve multi-node and symmetric multiprocessor (SMP) parallelism.

V. GPU Implementation

The massive parallelism and fast memory present in graphics processor hardware makes GPUs an obvious choice for many image processing tasks. The Compute Unified Device Architecture provides a convenient interface for programming image processing and other general-purpose computation tasks on NVIDIA GPU hardware [11]. The basic unit in the CUDA interface is the *thread*. Threads are organized into *blocks* that are mapped to GPU processors. The threads in a block share a limited but fast on-chip memory called *shared memory*. Threads in distinct blocks may not communicate. A large store of *global memory* provides slower access, and is often a staging area between CPU memory and the work happening in GPU multiprocessors. A complete review of the CUDA architecture is available in [11].

The process of calculating TPCF features contains both fine and coarse parallelisms: the computation of sample updates (fine) and computation of ROIs (coarse). The fine level of parallelism exists within a single ROI and is the computation of the select update values, the normalization of updated locations, and the bilinear interpolation to calculate $S_2^{(i)}(\mathbf{x})$. There are no dependencies in any of these processes so each is easily distributed. The coarse level of parallelism is the simultaneous calculation within multiple ROIs. Clearly the process of correlation updating is dependent though, as updating the autocorrelations for $\Phi_{x+1,y}$

requires the autocorrelations for $\Phi_{x,y}$ to be available. Both of these levels of parallelism can be effectively mapped to GPU using the CUDA block and thread parallelisms.

At the fine level, the computation of update values and bilinear interpolations can be divided among threads in a single block. Fine level details aside, at the course level sequences of dependent ROIs can be divided among blocks. The arrangement into dependent sequences of ROIs is essential since distinct blocks are unable to cooperate. A simple way to achieve this arrangement is for each block may process a horizontal strip of ROIs $\{\Phi_{0,y},\Phi_{1,y}, \ldots ,\Phi_{N-w,y}\}$. Each block may then perform the updating sequentially on its sequence of ROIs while other blocks do the same, achieving the coarse parallelism.

For the fine level parallelism the kernel runs in an iterative manner, starting with initialized values for $R^{(i)}_{0,y}$, $i = 1, 2, \ldots, P$ and lists of the sampling locations and interpolation coordinates. In the first step, portions of $\Phi_{0,y}$ are loaded into shared memory, and each thread calculates the update values for one sampling location until the list is exhausted. These sampling locations are then updated and normalized. In the next step, each thread then calculates one interpolation until the interpolation list is exhausted. The threads then reduce the interpolations, averaging to calculate S. The kernel repeats this process for each ROI in the dependent sequence and then expires.

A. Memory access patterns and shared memory

With each thread calculating a pair of update values, the memory accesses to Φ overlap significantly among threads. Each update value requires w multiply-add operations, and some elements of Φ may be accessed up to w times. For this reason Φ is stored in shared memory to avoid repeated reads to global memory. This decision is key since effective shared memory usage is one of the critical components of algorithm performance on GPU. Due to limited shared memory sizes, the autocorrelation matrices are maintained in global memory. This presents a problem as the calculation of update values cannot be organized among the threads so that accesses to $R^{(i)}$ are coalesced. Storing R in texture memory would be beneficial for caching and hardware interpolation, however textures are read only and cannot be changed within the duration of a kernel.

The limitations on shared memory size prohibit a general implementation for different ROI sizes. For this reason, the implementation used in Section VIII focused on the case $w = 32, \Delta\theta = \pi/8$ that is useful for the analysis of 5X magnification images. Each block was assigned 128 threads, 2176 bytes of shared memory, and 8 registers/thread to achieve a 100% occupancy on a Quadro FX5600 card.

VI. Related works

For the proposed fast correlation updating algorithm, a similar idea is found in the work on fast median filtering [12]. In this algorithm the ROI filter response is calculated at every position in the image by updating a kernel histogram based on the incoming and exiting information as the ROI shifts.

Commodity graphics hardware has become a cost-effective parallel platform to implement biomedical imaging applications. In radiology, GPU processing has been used to accelerate tomographic reconstruction [13], registration [14], and segmentation [15]. In pathology, GPU processing has been used to accelerate both nonrigid registration [16] and segmentation [17] of very large high-resolution microscopy images.

VII. Experimental Setup

Experiments were performed to examine the effects of correlation updating, parallelization, and GPU implementation. Four implementations for calculating TPCF features were produced:

1. **Serial direct FFT.** A fully serial implementation of the direct FFT-based method, written in C++ using the FFTW library [6].
2. **Serial correlation updating.** A fully serial implementation of the correlation updating method, written in C++.
3. **Parallel correlation updating.** A parallel SMP/multi-node implementation of the correlation updating method, written in C++ using MPI.
4. **GPU correlation updating.** A GPU implementation of correlation updating, using C++/CUDA. The implementation is specific for $w = 32, \Delta\theta = \pi/8$.

A. Hardware

The above implementations were tested on a GPU equipped cluster, the BALE system at the Ohio Supercomputer Center. The BALE supercomputer is endowed with 55 workstation nodes based on a dual-core Athlon 64 X2 architecture with integrated graphics card and 16 visualization nodes enhanced with dual-socket × dual-core AMD Opteron 2218 CPUs and dual-card Nvidia Quadro FX 5600 GPUs. All of these nodes are interconnected with Infiniband, and include a 750 GB, 7200 RPM local SATA II hard disk with 16 MB cache.

All GPU experiments and comparisons were run on the sixteen visualization nodes, where each node has 8 GB of DDR2 DRAM running at 667 MHz on the CPU side and 2×1.5 GB of on-board GDDR3 DRAM running at 1600 MHz on the GPU side, for a total of 11GB available DRAM per node. The remaining experiments were performed on the workstation nodes.

B. Data

Two sets of data were used in testing the three implementation varieties. The first dataset is used to compare direct-FFT and correlation updating and to examine scalability for the parallel implementation. It consists of ten 1000 × 1000 five-phase images taken from 20× magnification digitized mouse placenta tissue stained with hematoxylin and eosin.

The GPU time performance experiments used randomly generated images of size 256×256, 512×512, and 1024× 1024 with two, four, and eight-phase variations.

VIII. Results

A. Correlation Updating

To compare the performance of correlation updating with the direct-FFT method, TPCF features were calculated for the ten test images using the parameters of Table I. Parameters were chosen to reflect typical choices for the segmentation 5× and 20× magnifications, and also favorable and unfavorable FFT sizes. In the power of two cases the $2w-1$ DFT was padded to $2w$. The transforms for the non power of two cases were not padded. Justification for this choice is provided in Table II where it is clear that this padding would be detrimental in the $w = 130$ case, and would only help marginally in the $w = 34$ case.

The execution times for the serial direct-FFT and correlation updating calculations are presented in Figure 3. The average per-image execution times for direct-FFT are 1280, 11637, 43129, and 126489 seconds for the w =32, 34, 128, and 130 sizes respectively. The corresponding average times for correlation updating are 162, 178, 3474, and 3557 seconds. Overall the increase in execution times from the small window cases to the large window cases are considerable. From $w = 32$ to 128, the increase for direct-FFT is 34× where the corresponding increase for correlation updating is only 21x. There is a strong penalty with the direct-FFT implementation for non power of two cases, roughly a 10× increase for the small window sizes and 3× for the large. The correlation updating implementation does not suffer the same penalties with commensurate increases limited to 1.1× for the small window case.

The average speedup factors for correlation updating are presented in Table III. The speedup factors range between 8× and 67× depending on w. The larger speedup factors correspond to the non-power-of-two sizes due to the large penalty on FFT performance.

B. Parallelization

To demonstrate parallelization scalability the TPCF features were calculated for the large power of two case using the parallel implementation of correlation updating on 2, 4, 8, 16, 32, and 64 processors on 1, 2, 4, 8, 16, and 32 nodes. The execution times for these configurations are presented in Figure 4. Table IV contains the speedup factors for the parallel execution as compared to a fully serial single node implementation. These speedup factors are depicted graphically in Figure 5. There is a consistent reduction in execution time all the way through 16 processors, with more limited gains for the 32 and 64 processor cases, indicating that the unparallelized portions of execution account for a considerable portion of the total execution time. There is a relatively large amount of communication required for the worker nodes to report TPCF values to the head node, with each ROI generating $P(w/2+1)$ double-precision elements. In the case of a single 1000×1000 test image this corresponds to approximately 1.85 GBytes. Where increasing the number of nodes reduces the time spent in computation, the time spent in communication remains unchanged and the result on scalability is apparent.

C. GPU Implementation

To demonstrate execution time performance TPCF features were calculated using correlation updating implementations on both CPU and GPU for random images of size 256×256, 512×512, and 1024×1024 with two, four, and eight phases. The results from this experiment are presented in Table V. The corresponding speedup factors are presented in Table VI. All measures of execution time include communication and transfer of data between the CPU and GPU. For both CPU and GPU, execution time increases linearly with image size and the presence of additional phases, as expected. The speedup factor is greater for the more compute-intensive cases with larger image sizes and more phases, as the total amount of time spent in communication represents a smaller percentage of the total execution time. The kernel execution is interrupted by the CUDA watchdog timer in the case of 1024 × 1024 eight phase image. This is a feature of CUDA enabled to interrupt a kernel after a prescribed period to prevent a loss of graphics response for the user. The duration of the kernel depends on the sizes of the dependent sequences of ROIs, so to avoid watchdog timer interruptions the the sizes of these sequences must be limited based on the allowed kernel execution maximum.

Numerical Accuracy—The calculation of TPCF feature vectors is just one step in the segmentation procedure. After the features are calculated, they are subjected to dimensionality reduction prior to being clustered to form a segmentation. To demonstrate the effect on the end segmentation result, segmentations were generated for a digitized sample of hematoxlyin and eosin stained mouse placenta tissue to segment two major tissue layers (the labyrinth layer vs. the spongiotropoblast layer) using both double-precision CPU calculated features, and single-precision GPU calculated features (Figure 6). The segmentations produced were identical to the pixel indicating that the loss of precision in feature calculation has no impact on the outcome of the downstream analysis in this case.

IX. Discussion and Conclusions

TPCF features provide a method for the segmentation of histological images, however, this capability is accompanied by a significant computational burden. The direct-FFT method for deterministic TPCF calculation makes use of an efficient algorithmic staple, but execution time is strongly influenced by ROI size w as dictated by FFT transform size guidelines. The direct method also neglects the sparse autocorrelation sampling pattern and and close relationship between neighboring regions of interest resulting in significant amounts of wasted computation.

This paper discusses a novel method of correlation updating that uses the derived relationship between the autocorrelations of neighboring ROIs to update TPCF values rather than computing them from scratch. This method simultaneously addresses the considerations of wasted computation and ROI size sensitivity without compromising accuracy. Using the linearity of correlation, the autocorrelation calculations can be updated from one ROI to the next, rather than computed from scratch. Furthermore, performing these updates directly in the image domain permits the sampling locations to be selectively updated, and frees the

algorithm from the sensitivity to ROI size. The improvements of correlation updating result in a speedup from 8–67× over the direct-FFT method.

Both multi-node and GPU hardware solutions were pursued to further reduce execution time. The parallelization of feature calculations produces a speedup of up to 42× on 64 processors, reducing the total execution time for the set of ten 1000×1000 test images from 9.6 hours to just 13 minutes. General purpose GPU implementation of correlation updating provides a further 16× improvement over CPU, without compromising accuracy in segmentation results. This gain is impressive considering it is more than equivalent to using 16 processors on eight nodes, and puts performance within reach of end users who do not have access to production computing clusters.

Acknowledgments

This work is partially supported by NLM contract HHSN276201000585P and NIH grant R21 MH083264.

References

1 . Mosaliganti K, Janoos F, Irfanoglu O, Ridgeway R, Machiraju R, Huang K, Saltz J, Lenoe G, Ostrowski M. Tensor classification of N-point correlation function features for histology tissue segmentation. Medical Image Analysis. 2009; 13(1):156–166. [PubMed: 18762444]

2. Pan T, , Huang K. Virtual mouse placenta: Tissue layer segmentation. International Conference of the Engineering in Medicine and Biology Society; Jan 2005; 31123116

3. Ridgway R, , Irfanoglu O, , Machiraju R, , Huang K. Image segmentation with tensor-based classification of n-point correlation functions. Proceedings of the Microscopic Image Analysis with Applications in Biology (MIAAB) Workshop in MICCAI; Dec 2006;

4. Janoos F, , Irfanoglu MO, , Mosaliganti K, , Machiraju R, , Huang K, , Wenzel P, , de Bruin A, , Leone G. Multi-resolution image segmentation using the 2-point correlation functions. Proceedings of IEEE International Symposium on Biomedical Imaging; 2007; 300303

5. Cooper L, , Machiraju R, , Ha KS. Multi-resolution image segmentation using the 2-point correlation functions. Proceedings of Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA) in CVPR; 2010.

6. Frigo M, Johnson S. The design and implementation of fftw3. Proceedings of the IEEE. 2005; 93(2): 216–231.

7. Knudsen K, Bruton L. Recursive pruning of the 2d dft with 3d signal processing applications. Signal Processing, IEEE Transactions on. Mar; 1993 41(3):1340–1356.

8. Hu Z, Wan H. A novel generic fast fourier transform pruning technique and complexity analysis. Signal Processing, IEEE Transactions on. Jan; 2005 53(1):274–282.

9. Franchetti F, , Puschel M. Generating high performance pruned fft implementations. Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on; April 2009; 549552

10. Forum MP. Tech Rep Knoxville, TN, USA: 1994Mpi: A message-passing interface standard.

11. Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with cuda. Queue. Mar. 2008 6:40–53. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500.

12. Tang G, Yang G, Huang T. A fast two-dimensional median filtering algorithm. 1978:128–130.

13. Stone SS, , Haldar JP, , Tsao SC, , Hwu W-mW, Liang Z-P, , Sutton BP. Accelerating advanced mri reconstructions on gpus. Proceedings of the 5th conference on Computing frontiers, ser. CF '08; New York, NY, USA: ACM; 2008 261272 [Online]. Available: http://doi.acm.org/10.1145/1366230.1366276

14. Muyan-Ozcelik P, , Owens J, , Xia J, , Samant S. Fast deformable registration on the gpu: A cuda implementation of demons. Computational Sciences and Its Applications, 2008. ICCSA '08. International Conference on; 200830
15. Grady L, , Schiwietz T, , Aharon S, , Mnchen TU. Proceedings of MICCAI 2005 II Springer; 2005 Random walks for interactive organ segmentation in two and three dimensions: Implementation and validation; 773780
16. Ruiz A, Ujaldon M, Cooper L, Huang K. Non-rigid registration for large sets of microscopic images on graphics processors. J Signal Process Syst. Apr.2009 55 [Online]. Available: http://portal.acm.org/citation.cfm?id=1527246.1527249.
17. Ruiz A, , Kong J, , Ujaldon M, , Boyer K, , Saltz J, , Gurcan M. Pathological image segmentation for neuroblastoma using the gpu. Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on; May 2008. 296299

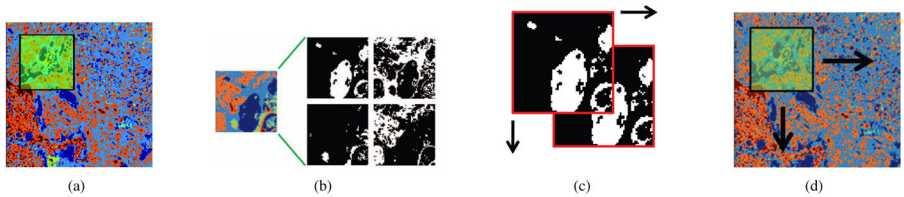


Figure 1.

Computation of TPCF features. (a) A ROI $\Phi(x, y)$ is defined in the phase image. (b) A binary mask is generated for each phase of the ROI. (c) The autocorrelation $R^{(i)}$ is calculated for each mask and normalized and sampled to generate the TPCF $S_2^{(i)}(r)$. (d) The ROI is iterated throughout the entire image.

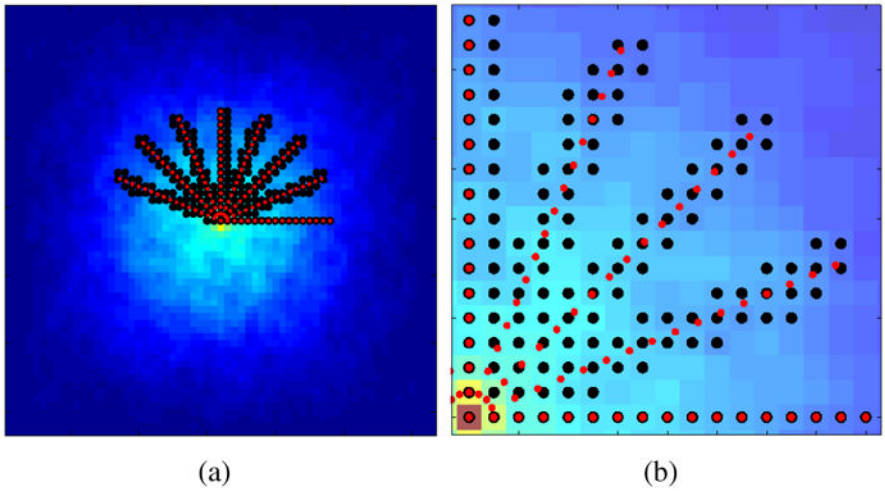
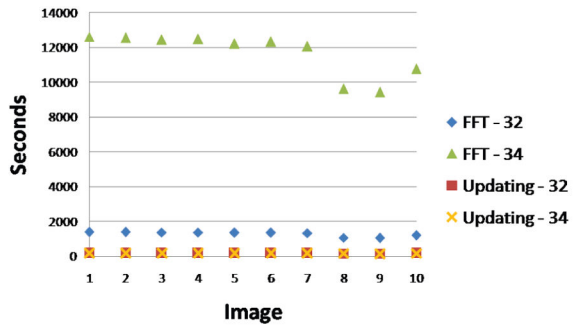
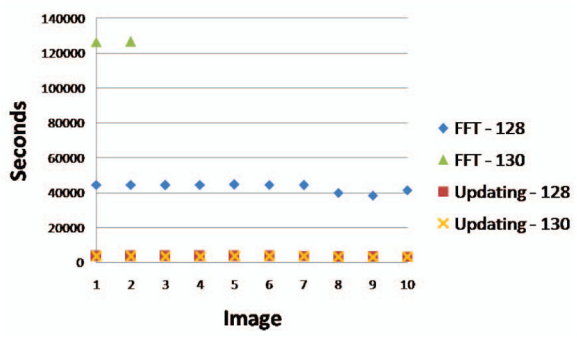


Figure 2.

Sparsity of samples for autocorrelation circumferential sampling. (a) Full autocorrelation matrix with the sampling pattern imposed. Red indicates the interpolation locations, black indicates bilinear sampling points. Very few samples of the full autocorrelation are required for calculation.



(a)



(b)

Figure 3.
Execution times for serial direct-FFT and correlation updating. (a) Small w case. (b) Large w case.

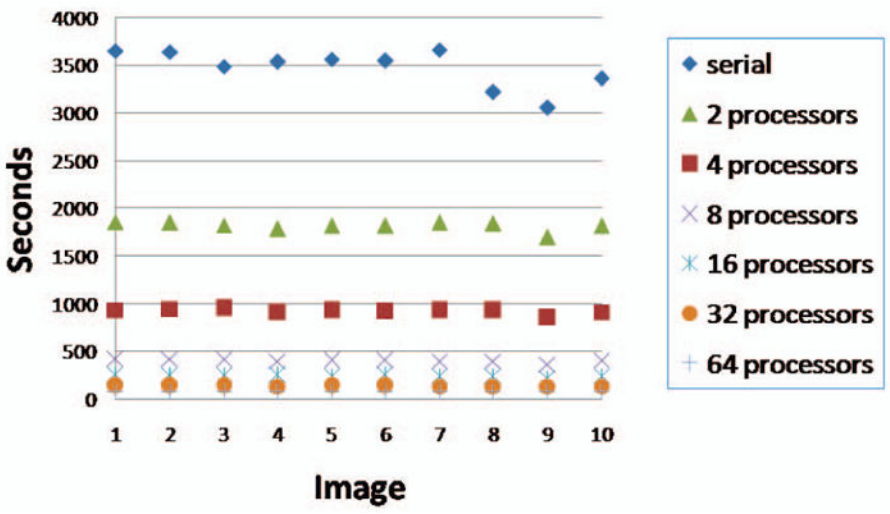


Figure 4.
Execution times for parallel correlation updating, $w = 128$ case.

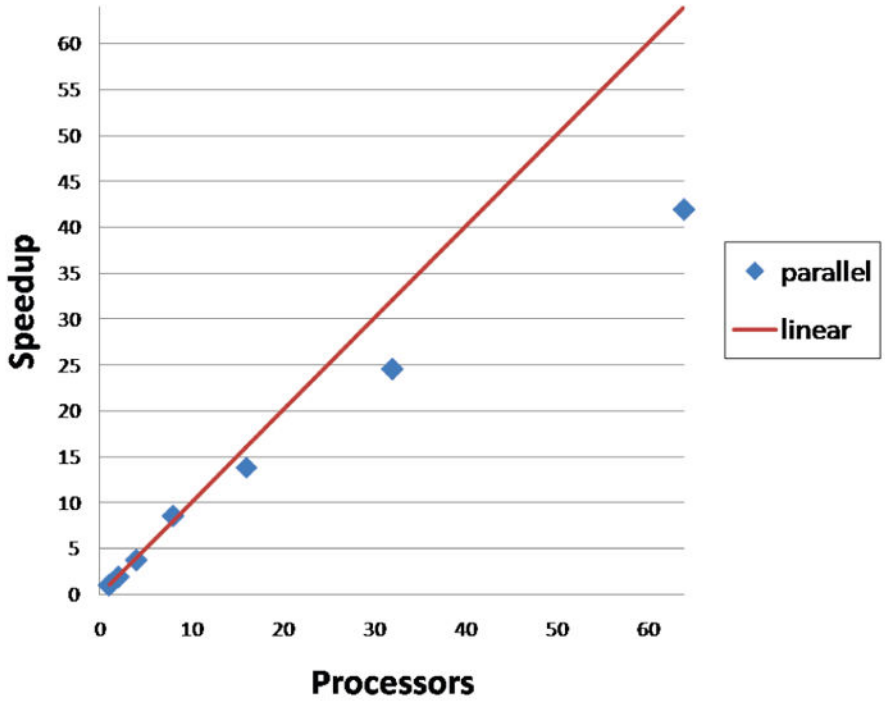


Figure 5.
Scalability of parallel TPCF correlation updating implementation.

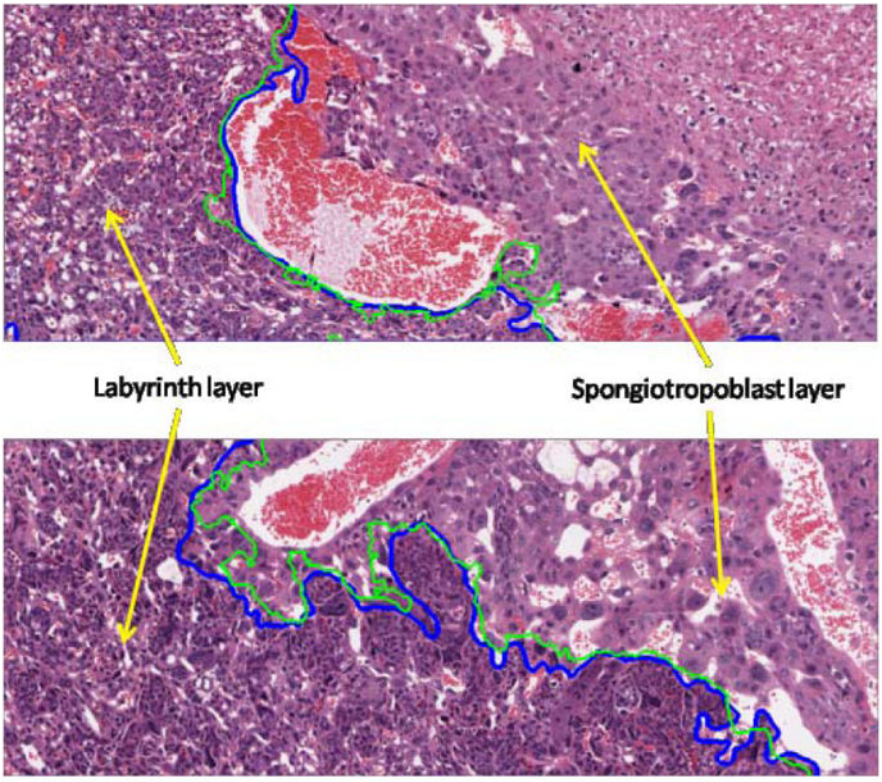


Figure 6.
Example results of segmentation of mouse placenta histology images. The green lines marks the boundary generated by the TPCF-based algorithm. The blue lines marks the boundary manually drawn by a student with pathology training.

Table I

Correlation updating and direct-FFT comparison parameters.

case	small-pow2	small	large-pow2	large
w	32	34	128	130
$\Delta\theta$	$\pi/8$	$\pi/8$	$\pi/16$	$\pi/16$

Table II

Padded DFT transform times. Averaged over 100 transforms.

w	32	34	64	128	130	256
milliseconds	0.27	2.6	1.9	10	31	53

Table III

Average speedup for correlation updating.

	Small		Large	
w	**32**	**34**	**128**	**130**
speedup	7.9×	67.0×	12.4×	35.6×

Table IV

Average speedup for parallel correlation updating, $w = 128$ case.

processors	**2**	**4**	**8**	**16**	**32**	**64**
speedup	1.9×	3.8×	8.5×	13.8×	24.5×	41.9×

Table V

Execution times for GPU correlation updating implementation.

	CPU, GPU		
phases	**256×256**	**512×512**	**1024×1024**
2	3.64, 0.33	16.69, 1.15	71.07, 4.52
4	7.28, 0.55	33.22, 2.21	141.13, 8.84
8	14.54, 1.02	66.41, 4.32	282.71, ★

★-watchdog timer intervention

Table VI

GPU/CPU speedup.

phases	256×256	512×512	1024×1024
2	11.1	15.6	15.7
4	13.2	15.0	16.0
8	14.2	15.4	★

★-watchdog timer intervention