Efficient Processing of Laser Speckle Contrast Images

W. James Tom*, Adrien Ponticorvo, and Andrew K. Dunn

***Abstract*—Though laser speckle contrast imaging enables the measurement of scattering particle dynamics with high temporal resolution, the subsequent processing has previously been much slower. In prior studies, generating a laser speckle contrast image required about 1 s to process a raw image potentially collected in 10 ms or less. In this paper, novel algorithms are described which are demonstrated to convert 291 raw images per second to laser speckle contrast images and as many as 410 laser speckle contrast images per second to relative correlation time images. As long as image processing occurs during image acquisition, these algorithms render processing time irrelevant in most circumstances and enable real-time imaging of blood flow dynamics.**

***Index Terms*—Biomedical image processing, blood flow, blood vessels, optical imaging, speckle.**

I. Introduction

LASER speckle contrast imaging is a technique useful for the characterization of scattering particle dynamics with high spatial and temporal resolution. Unfortunately, the image processing is slow. The advantages of laser speckle contrast imaging have created considerable interest in its application to the study of blood perfusion in tissues such as the retina [1] and the cerebral cortices [2]. In particular, functional activation [3] and spreading depolarizations [4]–[6] in the cerebral cortices have been explored using laser speckle contrast imaging. The high spatial and temporal resolution capabilities of laser speckle contrast imaging are incredibly useful for the study of surface perfusion in the cerebral cortices because perfusion varies between small regions of space and over short intervals of time. Regrettably, the processing of laser speckle contrast images has previously required about a second to process a frame while acquisition can occur at rates exceeding 100 frames per second. Consequently, the processing of laser speckle contrast images hinders complete use of the available temporal resolution. In this paper, algorithms will be described which are useful for laser speckle contrast imaging in general but emphasis will be to application of the algorithms to cerebral blood flow studies. These algorithms provide a dramatic improvement in processing times without any approximations.

Manuscript received April 2, 2008; revised April 21, 2008. Current version published November 21, 2008. This work was supported by the National Institutes of Health under Grant NS050150, in part by the American Heart Association under Grant 0735136N, and in part by the National Science Foundation under Grant CBET/0737731. *Asterisk indicates corresponding author.*

*W. J. Tom is with the Department of Biomedical Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: wjtom@mail.utexas.edu).

A. Ponticorvo and A. K. Dunn are with the Department of Biomedical Engineering, The University of Texas at Austin, Austin, TX 78712 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMI.2008.925081

Laser speckle contrast imaging also has the advantage of requiring a simple, inexpensive experimental apparatus. When collecting laser speckle contrast images, the coherent light of a laser illuminates a sample. Coherent light detected by a camera is time-integrated over the exposure duration to yield an image with a speckle pattern of variable amplitude dependent on motion and other parameters. Speckle amplitude may be quantified by computation of speckle contrast over a region of the image typically a square which will subsequently be referred to as a window. The length of the side of a square window in pixels will be represented by w. The width and height of a raw image in pixels will be represented by m and n, respectively, while the width and height of a processed image are $m' = m - w + 1$ and $n' = n - w + 1$, respectively. In (1) speckle contrast k is seen to be equal to the standard deviation of time-integrated intensity s_I divided by the mean time-integrated intensity $\langle I \rangle$

$$k = \frac{s_I}{\langle I \rangle} = \frac{\sqrt{\frac{\sum_{i=1}^{N}(I_i - \langle I \rangle)^2}{N-1}}}{\langle I \rangle}. \quad (1)$$

The time-integrated intensity measurements, I_1, I_2, I_3, ..., and I_N where N is a positive integer, may be from a single location at different times, multiple locations at one time, or multiple locations at different times. However, the set of samples must result from the same process. For example, in a functional activation experiment the temporal window over which samples are collected must be significantly less than the duration of the functional activation induced change in perfusion; including too many temporal samples will result in the undesirable situation in which the samples represent different physiological states. In the spatial sampling case, the spatial window must not be significantly greater than the diameters of the blood vessels of interest. Within these temporal and spatial sampling window limits, samples collected over time at one location are equivalent to samples from multiple locations at one time if ergodicity is valid. Typically, ergodicity is a reasonable assumption. Because there is generally more spatial resolution than is necessary while temporal resolution may be barely sufficient for studying cerebral blood flow dynamics, a spatial window of samples at a single point in time is typically used. For the spatial sampling used in the algorithms to be discussed $N = w^2$.

Direct application of (1) is slow. In [5], the authors note that processing the laser speckle contrast images required about 13 s per speckle contrast image including a minor portion of the time for additional processing, while in [4] the authors required 7.5 s to generate a laser speckle contrast image from 10 raw frames with collection occurring at 150 frames per second. Clearly, the processing is much slower than the rate at which images are acquired. Le *et al.* recommend using temporal sampling because they find that temporal sampling requires 0.34 s per speckle contrast image versus 2.51 s for spatial sampling [7]. Although temporal sampling may be satisfactory for studying relatively slow phenomena such as cortical spreading depression, sacrificing temporal resolution is not acceptable for functional activation studies. Forrester *et al.* developed an approximation of laser speckle contrast imaging which allowed processing to complete in 0.5 s per image [8]. He and Briers developed a computationally efficient spatial sampling algorithm [9], [10]. The algorithm of He and Briers has gone largely unnoticed which is unfortunate because it potentially offers a noticeable improvement over the most frequently used algorithms. In this paper, spatial sampling methods will be described which allow processing of laser speckle contrast images at orders of magnitude greater speed than currently used algorithms. Four different algorithms for computing speckle contrast will be considered: the direct algorithm, the direct algorithm using sums, a fast Fourier transform-based convolution algorithm, and a novel algorithm. After discussion of these four approaches, parallel algorithms will be considered.

Although speckle contrast provides information about the underlying process dynamics, it is usually further processed into a form that is approximately proportional to perfusion [3], [5]. First, speckle contrast is converted to correlation time which is a measure of the decay rate of the field autocorrelation. Then, the reciprocal of the correlation time is divided by the reciprocal of a baseline measure of correlation time that is derived from a different laser speckle contrast image. The ratio of the reciprocals of the correlation times for each pixel forms what will be referred to as a relative correlation time image. The values in the relative correlation time image have been found to be approximately proportional to the relative change in perfusion though the relationship to absolute perfusion is not easily defined [2], [11].

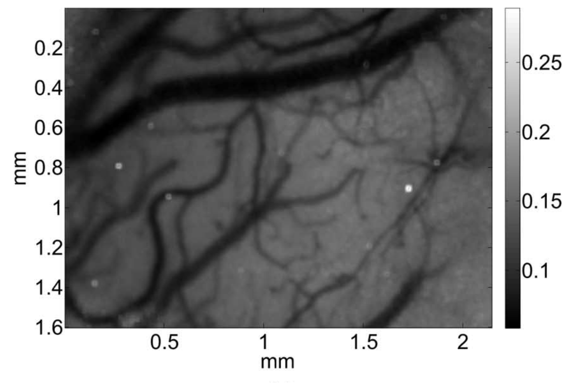
Generally, the velocities of the scattering particles are assumed to yield a Lorentzian spectrum which leads to (2) which relates speckle contrast to correlation time [12] where k is the speckle contrast, β is a constant which accounts for speckle averaging but is often neglected, and x is the exposure duration of the camera divided by the correlation time

$$k^2 = \beta\frac{\exp(-2x) - 1 + 2x}{2x^2}. \quad (2)$$

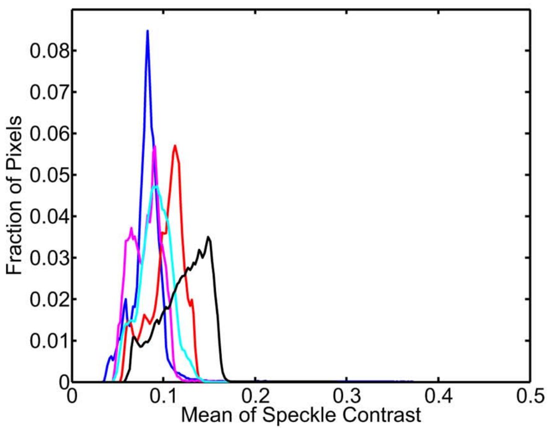
Since most experiments use a single exposure duration, computing the ratio of x from a given speckle contrast value to x from the baseline value is equivalent to the ratio of the reciprocals of the correlation times. Because the relationship between speckle contrast and correlation time is nonlinear, root-finding methods such as the Newton–Raphson method are often used [13]. Since such root-finding methods impose a significant computational burden, simpler methods have been explored [8], [13]. In the final part of this paper, different approaches to converting laser speckle contrast images to relative correlation time images will be discussed in an analogous manner to the treatment of the generation of laser speckle contrast images.

II. Typical Cerebral Blood Flow Measurements

Before describing any algorithms, typical results from rat cerebral blood flow studies will be analyzed to discover the computational precision necessary to process the images with fidelity. To expose the cerebral cortex, a craniotomy was performed on each experimental male Sprague–Dawley rat anesthetized with urethane (1.5 g/kg). All animal procedures were approved by the Animal Care and Use Committees of the University of Texas. The exposed portion of the cerebral cortex was illuminated with a 785-nm wavelength laser diode (Sanyo DL7140–201S). Images were captured with 5 ms exposures using an 8-bit CMOS camera (Basler A602f) operating at its maximum frame rate of 100 frames per second. Fig. 1(a) is a typical laser speckle contrast image of rat cerebral cortex. 300 raw images were processed using a 7×7 pixel window into 650×485 pixel laser speckle contrast images and were averaged. The blood vessels appear dark indicating a low speckle contrast value.



(a)



(b)

Fig. 1. (a) Example image of rat cerebral blood flow from 300 averaged laser speckle contrast images with windows of 7×7 pixels. (b) Histograms of speckle contrast over the entire laser speckle contrast image from five experiments each with a different rat. The black distribution is from the image in (a).

Fig. 1(b) shows the distributions of speckle contrast values from five distinct laser speckle contrast images produced from five different experiments using five different animals. Speckle contrast values were counted via placement into equally spaced bins with an interval of 0.002. As in Fig. 1(a), 300 laser speckle contrast images were averaged. The black distribution is from the image in Fig. 1(a). Most of the pixels have speckle contrast values below 0.15, and virtually no pixels have speckle contrasts above 0.2.

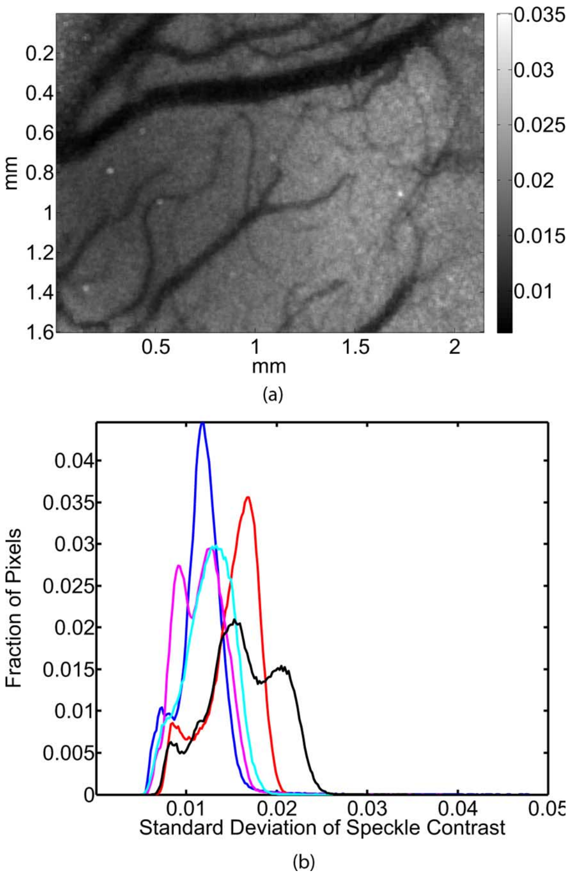


Fig. 2. (a) Image of the standard deviation of speckle contrast among the 300 laser speckle contrast images of Fig. 1(a). (b) Histograms of the standard deviation of speckle contrast among 300 laser speckle contrast images for five laser speckle contrast image sets. The black distribution is from the image in (a).

Fig. 2(a) is an image of the standard deviation of the speckle contrast values at a given pixel throughout the 300 laser speckle contrast images that contributed to the averaged laser speckle contrast image in Fig. 1(a). Thus, Fig. 2(a) is an estimate of the uncertainty in the speckle contrast values of Fig. 1(a). Generally, the areas representing the blood vessels have a lower standard deviation than the surrounding tissue indicating that the speckle contrast values from these regions are closer to converging to the true speckle contrast value than the surrounding tissue. This is expected because higher velocities have shorter correlation times which effectively leads to more independent samples of the process dynamics within a given exposure duration. With greater sampling, the variance is less, and the statistics converge faster.

The distributions of the standard deviations of speckle contrast at a given pixel are presented in Fig. 2(b). Equally spaced bins with an interval of 0.0002 were used. The distributions in Fig. 1(b) and Fig. 2(b) with the same color were generated from the same data. The five distributions show that the standard deviation may be as high as about 0.025 and as low as 0.005. Based on Fig. 1(b) and Fig. 2(b), the speckle contrast values of a single laser speckle contrast image are only about one order of magnitude greater than the noise floor in cerebral blood flow studies. These results are comparable to the variations in speckle contrast during repeated electrical forepaw stimulation in rats [14]. The vast majority of the noise is from variability in physiology. Because the signal-to-noise ratio is low, averaging of laser speckle contrast images is almost always performed. Using the smallest standard deviation from the five distributions leads to a standard error of 0.00029 when averaging 300 laser speckle contrast images. Consequently, 32-bit IEEE floating point numbers are sufficient to represent the speckle contrast values for these example measurements.

III. Speckle Contrast Algorithms

In this section, we describe four different algorithms for computation of laser speckle contrast images. A comparison and analysis of the performance of each algorithm including computational complexity follows the descriptions.

A. Direct Algorithm

In the direct algorithm, (1) is repeatedly evaluated as the window is slid across the image. First the mean of the time-integrated intensity values over the window is computed. Then the difference between each time-integrated intensity value of the window and the mean is squared and summed with the other squared differences. The final sum is divided by $w^2 - 1$, and the square root is evaluated. The final step is division by the mean. For image generation, the window is moved horizontally and vertically while repeating the process described here. Henceforth this method will be referred to as the "direct" method. He and Briers refer to this as the "naive" algorithm in [9]. Ignoring the memory required for the raw images and the laser speckle contrast images, memory consumption is $O(1)$. In future discussion as it has been here, the memory unavoidably consumed by the raw images and laser speckle contrast images will not be included in an algorithm's memory consumption to prevent obscuring algorithmic differences by such an inclusion.

B. Sums Algorithm

A minor modification of the direct algorithm is to express the standard deviation in an alternate form with sums as in

$$k = \frac{s_I}{\langle I \rangle} = \frac{\sqrt{\frac{N\sum_{i=1}^{N} I_i^2 - \left(\sum_{i=1}^{N} I_i\right)^2}{N(N-1)}}}{\frac{\sum_{i=1}^{N} I_i}{N}}. \tag{3}$$

First the sum of all of the time-integrated intensity values within the window is calculated followed by the sum of all of the squared time-integrated intensity values within the window. Then the sum of the squares is multiplied by w^2, and the sum squared is subtracted. The difference is divided by $w^2(w^2 - 1)$. After determining the square root, the result is divided by the mean. Note that the sums and the sums of squares of the time-integrated intensities quantized by the camera should be computed exactly using integer arithmetic to avoid low computational precision associated with (3) and finite precision arithmetic when the speckle contrast is small [15]. Computation of speckle contrast using this alternate formulation will be called the "sums" method, while He and Briers refer to this method as the "improved naive" algorithm [9]. Like the direct algorithm, memory consumption is $O(1)$.

C. FFT Algorithm

By considering the problem as a computation involving sums and sums of squares, more efficient methods may be derived which achieve greater efficiency by optimizing calculation of the sums and sums of squares. In fact, there are a considerable number of computations which are wasted whenever the sum, sum of the squares, and the mean is reinitialized when translating the window. One method to exploit the calculations from neighboring windows is to compute the sums and sums of the squares as a convolution with a square window with constant coefficients. The convolution may be performed in the frequency domain with a FFT.

First the FFT of the window is computed. The result may be computed before processing a batch of images with the same window and image size. Next the FFT of the image is complex multiplied by the FFT of the window element-by-element to achieve convolution. An inverse FFT yields an image which consists of the sums. The same process is performed on an image in which its elements have been squared. Computation of the speckle contrast values occurs via the same final steps as the sums method. Since using an FFT necessitates floating-point arithmetic for the sums and sums of squares, high precision floating-point arithmetic may be necessary to avoid computational inaccuracy in (3) [15]. Subsequent references to this method will be as the "FFT" method. Unlike the direct and sums method which require only stack space, in addition to stack space the FFT method requires two 2-D arrays of complex numbers and two 2-D arrays of real numbers all with dimensions equal to the dimensions of the raw image. Hence, the FFT method consumes $O(mn)$ memory.

D. Roll Algorithm

Although the FFT is an efficient algorithm, exploitation of the nature of the speckle contrast computation allows bypassing it for greater performance. Rather than perform convolution, vertical and horizontal rolling sums are computed for the image and a squared version of the image. For discussion, the vertical rolling sums will proceed from the top to the bottom of the image, while the horizontal rolling sums will be from the left to the right. First, the top w rows are summed as vectors to initialize the row of accumulated sums (Step 1). Then, the subsequent rows below are processed iteratively one at a time from the top to the bottom by vector subtraction of the row which is w rows above from the row of accumulated sums immediately followed by vector addition of the current row. The row of accumulated sums is stored once for each row below the top $w-1$ rows to produce a 2-D array with dimensions m and $n-w+1$ (Step 2). All further operations are performed on the new 2-D array resulting from this vertical rolling sums process. Next, the leftmost w columns are added as vectors to initialize the column of accumulated sums (Step 3). Then for each remaining column, the column which is w columns to the left is subtracted vectorially from the column of accumulated sums followed by vector addition of the current column in an iterative manner completely analogous to the process applied to the rows except proceeding from left to right rather than top to bottom. For all columns to the right of the leftmost $w-1$ columns, the column of accumulated sums overwrites the columns of the 2-D array containing the vertical rolling sums one column at a time proceeding sequentially from the leftmost column to the $m-w+1$ column which is furthest on the right (Step 4). The final 2-D array with effective dimensions of $m-w+1$ and $n-w+1$ contains the sums of time-integrated intensity over each possible window. Squared sums may be calculated by first squaring the values and using the same process. Finalization of the computation proceeds in the same manner as the two previously discussed algorithms (Step 5). This algorithm is easily generalized to rectangular windows. As with the sums algorithm, exact integer arithmetic is used for computing the sums and sums of squares to avoid computational inaccuracy in (3) [15]. This algorithm will be referred to as the "roll" method.

The pseudocode below implements the roll method as described above. $\overrightarrow{R_i(\mathbf{Image})}$ and $\overrightarrow{C_j(\mathbf{Image})}$ indicate the ith row vector of $\mathbf{Image}$ and the jth column vector of $\mathbf{Image}$ respectively. Operator symbols represent conventional vector operations. $\mathbf{Raw}$, $\mathbf{K}$, $\mathbf{Sum}$, and $\mathbf{SqSum}$ are 2-D arrays or images which represent the raw image, the speckle contrast image, the sum image, and the square sum image, respectively, while $\overrightarrow{SumAc}$ and $\overrightarrow{SqSumAc}$ are vector accumulators which hold the rolling sums and square sums, respectively. The parenthetical steps from the above description precede the pseudocode that implements the step, shown on the top of the next page.

The fast algorithm of He and Briers eliminates some of the redundant computations of the sums method but does not completely eliminate them as does the roll method because the fast algorithm of He and Briers performs rolling sums in only one dimension while the roll method performs rolling sums in two dimensions. Consequently, the algorithm of He and Briers is an $O(wmn)$ algorithm [9], while the roll method is an $O(mn)$ algorithm.

Though the above description of the roll method fully describes the efficiency of the approach in terms of arithmetic operation counts, it performs suboptimally on a modern computer. For maximal performance on a modern computer, as the vertical rolling sum is updated for a given pixel in a row the horizontal rolling sum is computed for that pixel. This process occurs simultaneously for both the sum and square sum calculations. As soon as the sum and square sum are available for a given pixel, conversion to speckle contrast occurs. Thus, the data flows efficiently among the processor's registers, arithmetic logic units, floating point units, and other digital circuits with considerably less use of the cache and significantly less likelihood of needing to access main memory which can have disastrous effects on computation time. Although either the vertical or horizontal rolling sums may be computed first, performance is best when traversal of the array occurs in such a manner that memory locations are accessed contiguously. Implementing the roll method in this manner requires two 1-D arrays of integers with length equal to m, the width of the raw image. Memory for stack space is also required but will generally represent an insignificant amount of memory as is the case for the three previously described algorithms. Therefore, memory consumption

```
{Step 1}
$\overrightarrow{SumAc} = \overrightarrow{R_1(\mathbf{Raw})}$
$\overrightarrow{SqSumAc} = \overrightarrow{R_1(\mathbf{Raw})} \cdot \overrightarrow{R_1(\mathbf{Raw})}$
for $i = 2$ to $w$ do
    $\overrightarrow{SumAc} = \overrightarrow{SumAc} + \overrightarrow{R_i(\mathbf{Raw})}$
    $\overrightarrow{SqSumAc} = \overrightarrow{SqSumAc} + \overrightarrow{R_i(\mathbf{Raw})} \cdot \overrightarrow{R_i(\mathbf{Raw})}$
end for
{Step 2}
$\overrightarrow{R_1(\mathbf{Sum})} = \overrightarrow{SumAc}$
$\overrightarrow{R_1(\mathbf{SqSum})} = \overrightarrow{SqSumAc}$
for $i = w + 1$ to $m$ do
    $\overrightarrow{SumAc} = \overrightarrow{SumAc} - \overrightarrow{R_{i-w}(\mathbf{Raw})} + \overrightarrow{R_i(\mathbf{Raw})}$
    $\overrightarrow{SqSumAc} = \overrightarrow{SqSumAc} - \overrightarrow{R_{i-w}(\mathbf{Raw})} \cdot \overrightarrow{R_{i-w}(\mathbf{Raw})} + \overrightarrow{R_i(\mathbf{Raw})} \cdot \overrightarrow{R_i(\mathbf{Raw})}$
    $\overrightarrow{R_{i-w+1}(\mathbf{Sum})} = \overrightarrow{SumAc}$
    $\overrightarrow{R_{i-w+1}(\mathbf{SqSum})} = \overrightarrow{SqSumAc}$
end for
{Step 3}
$\overrightarrow{SumAc} = \overrightarrow{C_1(\mathbf{Sum})}$
$\overrightarrow{SqSumAc} = \overrightarrow{C_1(\mathbf{SqSum})}$
for $j = 2$ to $w$ do
    $\overrightarrow{SumAc} = \overrightarrow{SumAc} + \overrightarrow{C_j(\mathbf{Sum})}$
    $\overrightarrow{SqSumAc} = \overrightarrow{SqSumAc} + \overrightarrow{C_j(\mathbf{SqSum})}$
end for
{Step 4}
for $j = w + 1$ to $n$ do
    $\overrightarrow{B} = \overrightarrow{SumAc}$
    $\overrightarrow{SqB} = \overrightarrow{SqSumAc}$
    $\overrightarrow{SumAc} = \overrightarrow{SumAc} - \overrightarrow{C_{j-w}(\mathbf{Sum})} + \overrightarrow{C_j(\mathbf{Sum})}$
    $\overrightarrow{SqSumAc} = \overrightarrow{SqSumAc} - \overrightarrow{C_{j-w}(\mathbf{SqSum})} + \overrightarrow{C_j(\mathbf{SqSum})}$
    $\overrightarrow{C_{j-w}(\mathbf{Sum})} = \overrightarrow{B}$
    $\overrightarrow{C_{j-w}(\mathbf{SqSum})} = \overrightarrow{SqB}$
end for
$\overrightarrow{C_{n-w+1}(\mathbf{Sum})} = \overrightarrow{SumAc}$
$\overrightarrow{C_{n-w+1}(\mathbf{SqSum})} = \overrightarrow{SqSumAc}$
{Step 5}
for $i = 1$ to $m - w + 1$ do
    for $j = 1$ to $n - w + 1$ do
        $\mathbf{K}_{ij} = \dfrac{\sqrt{\dfrac{w^2 \times \mathbf{SqSum}_{ij} - \mathbf{Sum}_{ij}^2}{w^2 \times (w^2 - 1)}}}{\dfrac{\mathbf{Sum}_{ij}}{w^2}}$
    end for
end for
```

is $O(m)$. Implementing the roll method in this manner is superior to naively implementing the algorithm according to the description above because it uses the resources of a modern computer more efficiently; there is no difference in arithmetic operation counts. Fewer memory operations are necessary, and the amount of computational work between memory operations is greater making it more likely that the processor will be able to exploit instruction level parallelism. Moreover, the long latencies in terms of processor cycles whenever memory is accessed and any limitations in memory bandwidth restrain the processor less. In addition, by using 1-D arrays instead of 2-D arrays, which would be required in a naive implementation of the roll method, the intermediate results are more likely to be accessible in the processor cache.

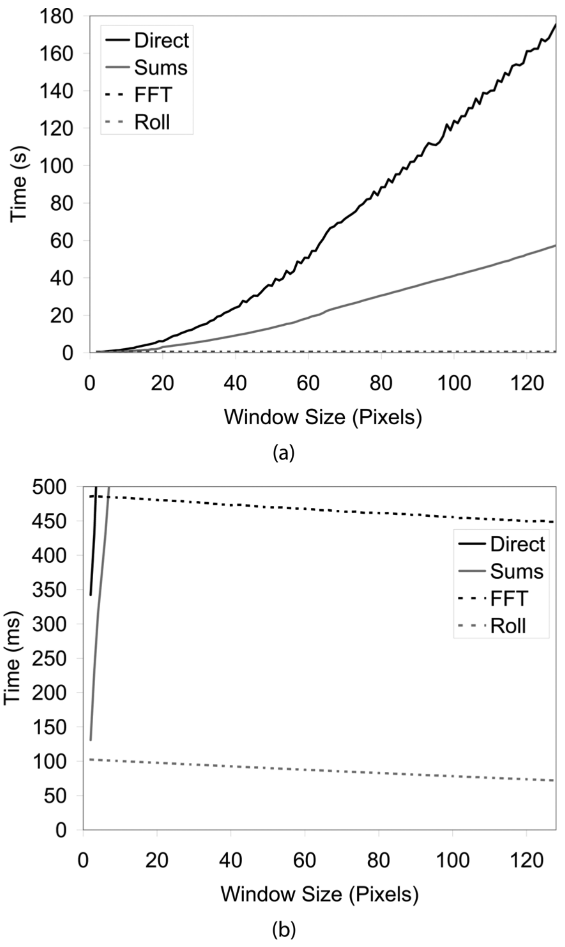


Fig. 3. (a) Average time in seconds to process 10 raw images with dimensions of 768×512 as the square window is varied from 2×2 to 128×128 pixels. (b) Magnified version of (a) in milliseconds instead of seconds. Each measurement was repeated 10 times. The standard error of the mean was always less than 161, 72, 0.91, and 0.15 ms for the direct, sums, FFT, and roll algorithms, respectively.

E. Performance Comparison

To evaluate the performance of the described algorithms, the algorithms were written in the C programming language and were compiled using Microsoft Visual C++.NET 2003. The implementations were limited to the ISO C standard. The binaries were executed within Microsoft Windows XP SP2 on an Intel Pentium D operating at 3.2 GHz with 2 GB of dual channel 533 MHz DDR2 RAM. All FFTs were performed with FFTW version 3.1.2.

Fig. 3(a) shows the computation time required to process 10 raw images of 768×512 pixels while varying the square window from 2×2 to 128×128 pixels. Fig. 3(a) shows that the computation time of the direct and sums method initially increases quadratically as the window size increases. If the window size was incremented until it equaled the size of the image, the rate of increase would slow until reaching a

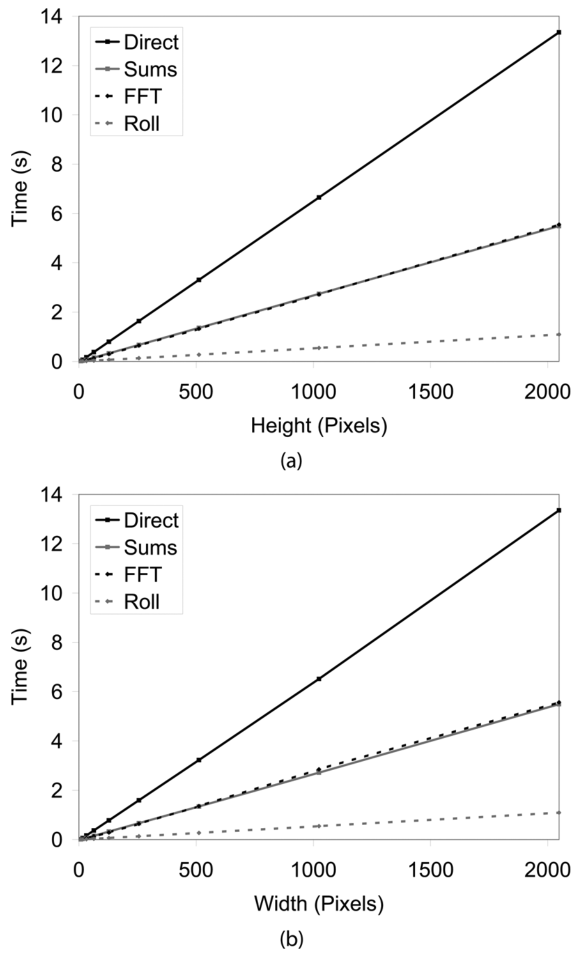


Fig. 4. (a) Average time in seconds to process 10 raw images 2048 pixels wide with a window size of 7 × 7 pixels as height is incremented from 2 to 2048 pixels by powers of 2. The sums and FFT performance curves overlap. (b) Average time in seconds to process 10 raw images 2048 pixels high with a window size of 7 × 7 pixels as width is incremented from 2 to 2048 pixels by powers of 2. The sums and FFT performance curves overlap. Each measurement was repeated 10 times, and the standard error of the mean was always less than 12.2 ms.

maximum when the window size is half of the raw image dimensions. Then the computation time of the direct and sums algorithms would decrease producing a nearly symmetric performance curve. In Fig. 3(a), the performance of the FFT and roll methods appears to be independent of window size. However, Fig. 3(b) which is a magnified version of Fig. 3(a) shows the gradual decrease in computation time as the window size is increased. The sums and roll method share the curve shape of the direct and FFT algorithms respectively but have lower overhead. The superior efficiency of the roll method is challenged only when the window consumes the entire raw image.

Fig. 4(a) and (b) addresses the impact of varying the height and width of raw images on computation time. The window size was fixed at 7×7 pixels, and either the width or height, whichever was not varied, was always 2048 pixels. Computation time represents processing of 10 frames. Only heights and widths which are powers of two are shown to reduce the complexity of analyzing the performance of the FFT algorithm. As shown, the direct, sums, and roll method have a linear relationship with image height and width. The FFT curves do not show much curvature despite the FFT having $n\log(n)$ scaling. Although linear scaling apparently dominates the performance curve for the FFT method, other problems arise when using this method with large images. Namely, with large images the inverse FFT has a tendency to overflow especially when small windows which have high spatial frequency bandwidths are used. For the other algorithms, overflow is only an issue when using impractically large windows. For overflow to even be possible, the window would have to exceed 181×181 pixels with 8-bit data with the implementations of the sums and roll algorithms of this paper. Furthermore, the window size would have to be larger in order for overflow to be likely. Though criteria for overflow in the case of the direct method is not easily defined because of the combination of integer and floating-point operations, experience indicates that the direct method will at least scale to window sizes possible with the sums and roll methods.

The measurements presented in Figs. 3 and 4 combined with theoretical analysis lead to (4)–(7) which describe the computation times per frame, T_d, T_s, T_f, and T_r, for the direct, sums, FFT, and roll algorithms, respectively, in terms of window size, raw image height, and raw image width. The nonnegative constants k_{d1}, k_{d2}, k_{s1}, k_{s2}, k_{f1}, k_{f2}, k_{f3}, k_{f4}, k_{r1}, k_{r2}, and k_{r3} will depend on how quickly the algorithm implementation executes on a given computer. In other words, the nonnegative constants are determined by the speed of addition, multiplication, memory loads and stores, and other instructions in the sequence of the implementation. For our test case, k_{d1} is 3.7 ns, k_{d2} is 220 ns, k_{s1} is 2.6 ns, k_{s2} is 190 ns, k_{f1} is 6.8 ns, k_{f2} is 6.8 ns, k_{f3} is 0 ns, k_{f4} is 26 ns, k_{r1} is 4.9 ns, k_{r2} is 3.2 ns, and k_{r3} is 18 ns as determined by linear least-squares fitting of the performance curves. The $\log(m)$ and $\log(n)$ in (7) may be reduced somewhat with a more complicated overlap-add or overlap-save method of FFT-based convolution

$$\begin{aligned} T_d &= k_{d1}w^2(m-w+1)(n-w+1) \\ &\quad + k_{d2}(m-w+1)(n-w+1) \qquad (4)\\ T_s &= k_{s1}w^2(m-w+1)(n-w+1) \\ &\quad + k_{s2}(m-w+1)(n-w+1) \qquad (5)\\ T_f &= k_{f1}nm\log(m) + k_{f2}mn\log(n) + k_{f3}mn \\ &\quad + k_{f4}(m-w+1)(n-w+1) \qquad (6)\\ T_r &= k_{r1}mn + k_{r2}m(n-w+1) \\ &\quad + k_{r3}(m-w+1)(n-w+1). \qquad (7)\end{aligned}$$

When setting height, width, and window size to values representative of an actual experimental setup, the roll method performs best followed by the FFT method as shown in Fig. 5. The direct method requires the most computation time. For the test, ten speckle contrast images were averaged. The raw image height was 512, and the width was 768. A window size of 7×7 pixels was used. The results represent the average computation time over 100 executions of each algorithm.

IV. Parallel Speckle Contrast Algorithms

Because the roll algorithm eliminates all arithmetic redundancy, algorithmic performance improvements are only possible

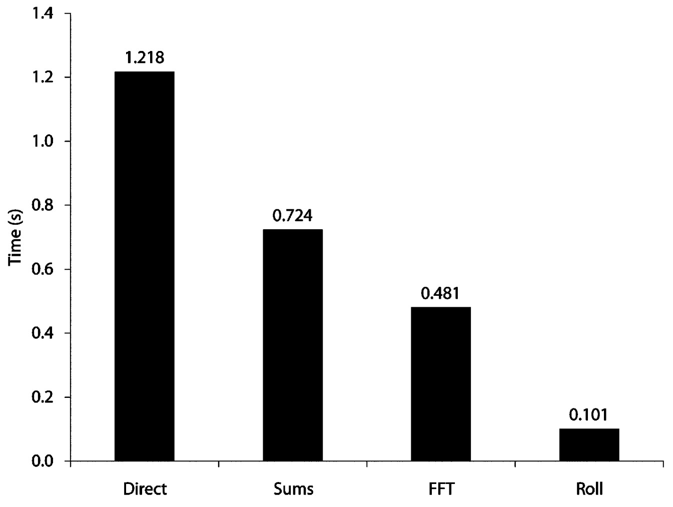


Fig. 5. Average computation times in seconds of laser speckle contrast algorithms with 10 raw images sized 768 × 512 and a window size of 7 × 7 pixels which represents typical experimental conditions. Each measurement was repeated 100 times, and the standard error of the mean was always less than 1.1 ms.

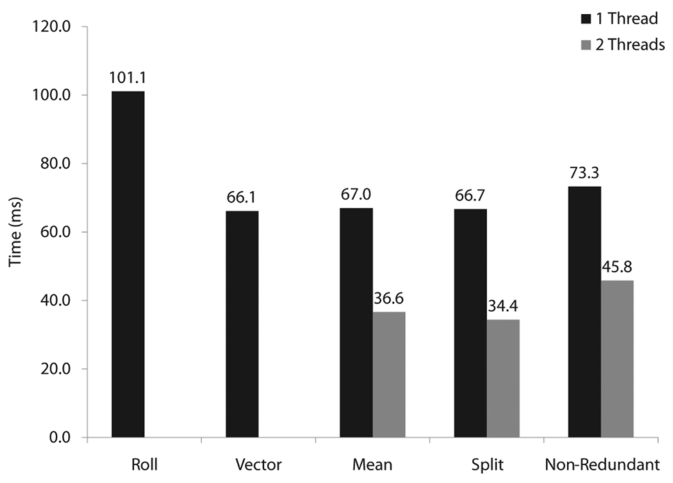


Fig. 6. Average computation times in milliseconds of vectorized and threaded laser speckle contrast algorithms with 10 raw images sized 768 × 512 and a window size of 7 × 7 pixels which represents typical experimental conditions. The split multithreaded algorithm yields the fastest computation time of 34.4 ms which is equivalent to 291 frames per second. Each measurement was repeated 100 times, and the standard error of the mean was always less than 0.10 ms.

via parallelism. Furthermore, since the roll method is superior to the other methods both theoretically and in the presented performance data for all variations of raw image and window size except when raw image and window size are equal, parallelism will only be considered for the roll method. With the roll method, multiple raw images may be simultaneously processed, a raw image may be partitioned, the vertical rolling sums along the column vectors may be computed concurrently, the horizontal rolling sums along the row vectors may be processed in parallel, and (3) may be evaluated simultaneously for different pixels. Each avenue for parallelism has caveats. Processing multiple raw images simultaneously introduces extra computations when averaging the speckle contrast images. Partitioning images requires overlap regions between the partitions which require extra calculations. Transitioning from parallel computation of the vertical rolling sums to parallel computation of the horizontal rolling sums is delayed by the necessary synchronization though the arithmetic operation burden is not greater than a nonparallel implementation of the roll algorithm. In this section, we demonstrate that parallel processing via vector operations and multithreading lead to an additional performance improvement resulting in processing speeds close to 300 frames per second with standard computing hardware.

Vector operations or single instruction multiple data (SIMD) operations available on many modern processor architectures allow the simultaneous application of a given arithmetic operation on a vector of data as opposed to scalar data. An implementation of the roll method using SIMD operations for the floating point calculations and to a lesser degree the integer computations will be demonstrated. This implementation of the roll method will be called "vector." For this implementation of the roll method, we used Streaming SIMD Extensions (SSE) and Streaming SIMD Extensions 2 (SSE2) supported by processors produced by Advanced Micro Devices (AMD) and Intel because the ISO C standard does not include explicit support for SIMD operations.

Fig. 6 shows the significant performance improvement between the vector implementation and the standard roll method. Image parameters in Fig. 6 are the same as in Fig. 5. Almost the entire 34.6% improvement is due to acceleration of the floating point calculations in (3). Little if any performance improvement is seen in vectorizing the integer operations because the processor used here automatically identifies and concurrently evaluates the integer operations which do not exhibit data dependence. A version of square root precise to only 11 bit was used and is acceptable at least in the application of laser speckle contrast imaging to cerebral blood flow measurements as indicated in the sample data presented earlier. Optimization of square root evaluation resulted in the largest performance improvement attributable to a single operation. Of course if there is interest only in the correlation time and not the speckle contrast, no square root is necessary. The SIMD operations used in the vector implementation were also employed in the three multithreaded approaches which will be described. Thus, in the ideal case, execution of the multithreaded algorithms with only one thread should require the same amount of time as the implementation entitled vector. Threading was implemented using the native threading functions of Microsoft Windows.

The simplest approach to parallelism via multithreading is to process distinct frames simultaneously to speed generation of an averaged laser speckle contrast image. This approach will be referred to as the "mean" threading method. Fast image averaging is performed by accumulating a sum at each pixel after conversion of each raw image to a laser speckle contrast image. After processing all the images to be averaged, each pixel is divided by the image count. In the mean approach fast image averaging can be used within the set of images dedicated to each thread, but after completion of each thread an additional averaging step is necessary among the sets of laser speckle contrast images derived from different threads. Also, the mean method will require $t-1$ additional 2-D arrays compared to the single-threaded implementation for storing the laser speckle contrast images for the t threads prior to the final averaging step.

Another multithreaded algorithm is to divide the raw image into segments and process the segments simultaneously. In this paper, this threading approach will be the "split" method, and the raw images will be split horizontally into equal numbers of adjacent rows. Because conversion of a raw image to a laser speckle contrast image results in a reduction of the horizontal and vertical dimensions, partitioning a raw image before processing will result in an image in which segments in the middle of the laser speckle contrast image are absent unless at the image division stage an overlap region between adjacent divisions is included. This method requires $t-1$ pairs of 1-D integer arrays of length equal to the raw image width beyond the memory requirements of the single-threaded case. The split method is an intermediate between the roll and sums algorithms. When one thread is used, the split method is identical to the roll algorithm. When $(m-w+1)(n-w+1)$ threads, the maximum possible number of threads, are used with the necessary vertical and horizontal segmentation, the split method is equivalent to the sums algorithm.

Single-threaded execution of the mean and split methods is nearly as fast as the vector implementation as seen in Fig. 6, but the overhead imposed by enabling threading is still observable. With the given height, width, and window size, the computation time reduction from increasing from one to two thread execution with the mean and split methods is 45.3% and 48.5%, respectively. In this test, the split method appears faster because there are about $(t-1)(w-1)mf/t$ redundant integer additions per thread which requires less time than about $(t-1)(m-w+1)(n-w+1)/t$ redundant floating point additions per thread of the mean method. f represents the number of averaged laser speckle contrast images or frames. In Fig. 6, the extra floating point additions of the mean method are more numerous than the extra integer additions of the split method and are individually more computationally expensive than an integer addition though only marginally so due to efficient use of SIMD instructions.

The final multithreading method exploits the independence of the computations involving the vertical rolling sums among columns and the horizontal rolling sums among rows to avoid the redundant calculations inherent to the other two methods. In this method which will be called the "non-redundant" method, rather than processing one row at a time, t rows are divided into t sets of adjacent columns over which t threads compute the vertical rolling sums on the shortened rows contained within each thread's assigned columns. Then, the horizontal rolling sums are computed for each of the t rows by one of the t threads. After using the sums and square sums to derive the speckle contrast at each pixel, the next t rows are processed by the same procedure until all rows in the raw image are processed. The non-redundant method needs two 2-D arrays of integers with dimensions of the raw image width by the number of threads rather than the two 1-D integer arrays of length equal to the raw image width as needed in the single-threaded case.

Although the non-redundant threading method has lower arithmetic operation counts as compared to the mean and split methods, deviation from the sequence of operations of the single-threaded standard C implementation and the frequent data synchronization requirements limit both the one and two threaded cases of the non-redundant method. Because the non-redundant method does not compute the horizontal sum of a row immediately following the vertical sum, less computational work is available between memory accesses. This explains the increase in computation time as compared to the vector implementation observable in Fig. 6. Moreover, burden imposed by the large amount of processor core communication for data synchronization limits the improvement from one to two threaded execution to 37.5% rather than the idealized 50%. Though this method performs poorly here, it will be the algorithm of choice for highly multithreaded processor architectures such as a graphics processing unit. In such architectures, instruction level parallelism matters less, while reduction of redundancy is very important for good performance.

V. Relative Correlation Time Algorithms

This paper will demonstrate four algorithms for conversion of a laser speckle contrast image to a relative correlation time image. All of the methods except for one are in some way dependent on a nonlinear root-finding method such as the Newton–Raphson method. Therefore, the most obvious image conversion method is to use the Newton–Raphson method at each pixel of a laser speckle contrast image. Subsequent references to this approach will be as the "Newton" method. A potentially more efficient method is to generate a table which stores the ratios of camera exposure duration to correlation time indexed by speckle contrast values. By using the index within the table which is closest to the measured speckle contrast value, a measure of the correlation time will be obtained which has bounded error. This will be called the "table" method. In the case that the desired precision necessitates a table which would be prohibitively large, a "hybrid" method may be used which uses a relatively small table to generate good guesses for the Newton–Raphson method so that solution convergence occurs much faster. The hybrid method demonstrated in this paper will involve one table lookup and a single iteration of the Newton–Raphson method. Previously, it has been noted that when the exposure duration is significantly greater than the correlation time an asymptotic approximation may be used [12], [13]. When the asymptotic approximation is valid, the relationship between speckle contrast, k, and x, the camera's exposure duration divided by the correlation time is described by (8). This approach will be called the "asymptote" method

$$x = \frac{1}{k^2}. \tag{8}$$

Fig. 7 shows the time required to convert laser speckle contrast images to relative correlation time images. The values represent time required to convert a single speckle contrast image with dimensions 762×506 which are the dimensions of speckle contrast images produced in the tests represented by Figs. 5 and 6. Each speckle contrast value was a randomly selected value between zero and one, the minimum and maximum theoretical speckle contrast values respectively. Each algorithm was executed in such a way as to give accuracy levels which were as equivalent as possible. A table containing 16384 values was used for the table method, while the hybrid method used 8192 values. The Newton method is about 100 times slower

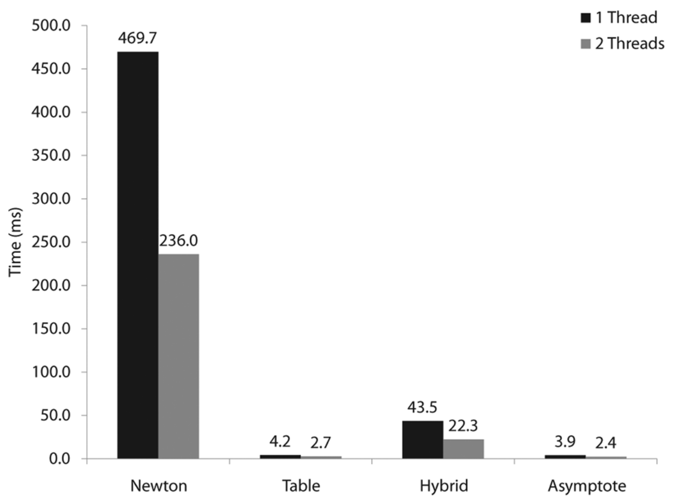


Fig. 7. Average computation times in milliseconds of relative correlation time algorithms with a laser speckle contrast image sized 762 × 506 which represents typical experimental conditions. Each measurement was repeated 100 times. For the Newton algorithm, the standard error of the mean was always less than 0.14 ms, while the standard error of mean was always less than 0.04 ms for the other algorithms.

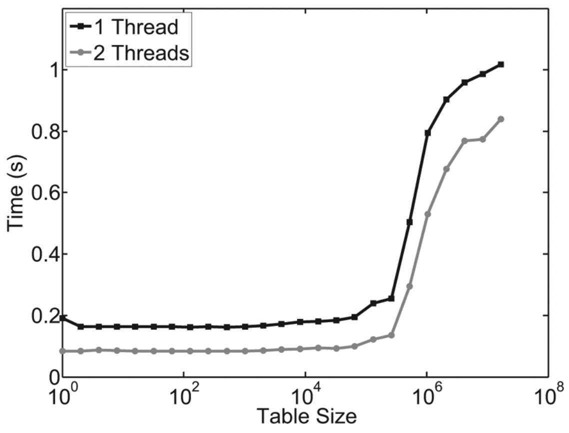


Fig. 8. Average computation time in seconds of the table method with a laser speckle contrast image sized 4096 × 4096 as table size is varied. Each measurement was repeated 100 times, and the standard error of the mean was always less than 0.33 ms.

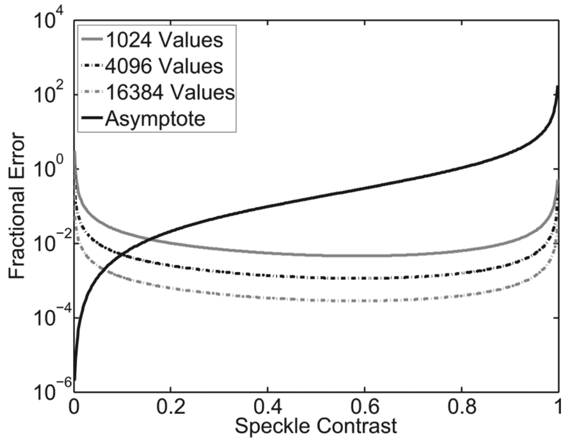


Fig. 9. Fractional error in determining the ratio of exposure duration to correlation time by the table and asymptote methods as speckle contrast is varied. The curves labeled “1024 Values,” “4096 Values,” and “16384 Values” represent the maximum fractional error for the table method with a table size of 1024, 4096, and 16384 values, respectively.

than the table and asymptote method while being about 10 times slower than the hybrid method. Also, Fig. 7 shows the thread scaling of each algorithm. When converting speckle contrast to correlation time, the pixels exhibit complete independence and share no calculations unlike the case for generation of speckle contrast images. Thus, processing relative correlation time images is readily parallelized and doubling the number of threads should nearly halve the computation time which is seen with the Newton and the hybrid methods. The table and asymptote methods deviate from this expected behavior because the computation time is so short that the thread initialization time imposed by the operating system is significant.

In Fig. 8, the effect of varying the table size on computation time of the table method is shown. 4096 × 4096 sized images were used to minimize the thread scaling impact of the operating system’s thread initialization time. Since the speckle contrast values are randomly selected, temporal and spatial locality optimizations in the processor cache and data prefetching are largely futile. Hence, the processor cache is effective only when the entire table fits within the cache. When the table is too large to fit within the cache, performance is limited by the latency and bandwidth of the processor’s memory system. Each processor core of the Intel Pentium D used in this paper has a two megabyte L2 cache. Each entry in the lookup table consumes 4 bytes. With real images, the values of neighboring pixels will likely be close and hence temporal and spatial locality within the lookup table will be exploitable. Furthermore, the error measurements from above indicate that actual data does not span the entire interval between zero and one as suggested by theory. With real data, the effective table size is the portion of the table actually accessed. Thus, with the data presented in the error measurements section the effective table size will be approximately a fifth of the actual size. Consequently, the results for the table method in Figs. 7 and 8 represent worst case scenarios. Because the memory system of each processor core is not independent, thread scaling deviates from ideal for large table sizes as seen in Fig. 8. Thread scaling is nearly ideal for small tables which fit within the L2 cache because each processor core of the Intel Pentium D used in this paper has its own L2 cache. Large table thread scaling is affected because both processor cores share the bus to main memory, and hence the latency and bandwidth issues when accessing main memory are exacerbated.

Fig. 9 shows the fractional error in determining the ratio of exposure duration to correlation time with the table and asymptote methods as a function of speckle contrast. Fractional error is the absolute value of the difference between the true value and the estimate value divided by the true value. Because generation of relative correlation time images involves ratios of x, fractional error is the best manner to quantify the significance of the deviation from the true value. The Newton–Raphson method was used to determine the true value of x. Clearly, for the asymptote method the deviation from the true value is unacceptable for high values of speckle contrast. However, with real data as shown in the sample experimental data the speckle contrast values are low. Thus, when there is caution, the asymptote method may be used.

Also shown in Fig. 9 is the maximum fractional error of the table method with table sizes of 1024, 4096, and 16384 values. Because the implementation of the table method in this paper chooses the index which is closest but not greater than the measured speckle contrast value, the error increases until the measured speckle contrast value becomes equal to the value of the next index. Therefore, the fractional error but not the maximum fractional error of the table method is a form of sawtooth wave with the minimum being zero. The maximum fractional error is the function which connects all of the peaks of the sawtooth wave. For a speckle contrast value k with $k_n \leq k < k_{n+1}$ where k_n and k_{n+1} are consecutive table indices, the maximum fractional error in x given k is $x_n/x_{n+1} - 1$ where x_n and x_{n+1} are the values indexed by k_n and k_{n+1} respectively. In Fig. 9 the maximum fractional error curve decreases as the table size increases or equivalently the interval between speckle contrast value indices decreases. Unfortunately, the fractional error of the table method increases significantly for very small and large speckle contrast values because the ratio of exposure duration to correlation time in (2) rapidly ascends to infinity for speckle contrast values that approach zero and rapidly descends to zero for speckle contrast values that approach one. Though the maximum fractional error curve is determined by the table size, the magnitude of the fractional error as the speckle contrast value approaches zero or one is affected little by table size.

VI. Conclusion

In this paper, laser speckle contrast image processing has been demonstrated at 291 frames per second. With the table method, relative correlation time image processing occurs at 375 frames per second, and when the asymptote method is applicable, it will deliver 410 frames per second. Consequently, the methods described in this paper allow real-time processing of raw images into relative correlation time images.

In order to obtain similar performance results, the roll method is recommended over the direct, sums, and FFT methods in all situations except when only a single speckle contrast value is needed. In this case, the sums method is recommended. Whenever possible, SIMD instructions and threading should be used. In most situations, the mean and split threading approaches will be about equally appropriate. Yet, when large amounts of image averaging is to be performed or when the window is very large the mean method will be preferable. Redundant calculations will prevent the mean and split threading methods from scaling well with high thread counts. The non-redundant method performed poorly in these tests because it is not suited to a processor which derives the majority of its performance from fast single-threaded execution. It is expected that with a highly multithreaded architecture such as a graphics processing unit the non-redundant method will excel.

When the speckle contrast values are small, the asymptote method of speckle contrast to correlation time conversion is preferable both for performance and accuracy reasons. Moreover, performance of the implementation used here could be further improved through use of SIMD instructions. The asymptote method is useful in more experiments than may be expected. In fact, β has been successfully ignored in many previous biological studies because the effect of β cancels when computing the relative correlation time when the asymptotic approximation is valid. For speckle contrast values which are too large for accurate application of the asymptote method, the lookup table method is recommended. The presented range of speckle contrast error in cerebral blood flow imaging suggests that a table containing 3500 values should be sufficient to find the correlation time to within the uncertainty of the measurement in typical cerebral blood flow studies. Using a larger table as was done in this paper allows more accurate determination of the most likely correlation time within the distribution of probable correlation times for a given speckle contrast measurement. Generally, only experience will lead to knowledge about the range of expected speckle contrast values because the correlation time depends on the measured process rather than known quantities such as the coherence time of the laser. Furthermore, increasing exposure duration of the camera to ensure low speckle contrast values is ill advised not only because it will decrease temporal resolution but also because of the decrease in image contrast as described by Yuan [14]. Consequently, we recommend implementation of both the asymptote and table methods and to select the appropriate method at computation time by the speckle contrast value. This enables more accurate conversion than either method alone with performance generally between that of the asymptote and table methods. The computation time is less than that of the table method in many cases because table size can be reduced while maintaining the same accuracy, and the asymptote method is slightly faster whenever arithmetic performance is greater than memory speed. If additional precision is necessary, then the results of either the asymptote or table methods, whichever is applicable, should be used as a good initial guess for an iterative nonlinear equation solver as was done in the hybrid method.

Besides the obvious benefit of being able to observe the progress of experiments during the experiment, the roll algorithm and its parallel variants described here make it practical to work with large data sets. For measuring processes such as functional activation which have low signal-to-noise ratio, the algorithms described here allow averaging of a greater number of experimental trials. When measuring a slow process such as cortical spreading depression, the algorithms described here facilitate working with data sets with increased temporal resolution. Because it is possible for speckle contrast processing to occur more quickly than data acquisition in most experimental situations, processing time is rendered irrelevant if processing is performed during acquisition by the algorithms described here. Implementations of the efficient algorithms described here for processing of laser speckle contrast images and relative correlation time images are available for download[1].

References

[1] J. D. Briers and A. F. Fercher, "Retinal blood-flow visualization by means of laser speckle photography," *Invest. Ophthalmol. Vis. Sci.*, vol. 22, pp. 255–259, 1982.

[1]http://www.bme.utexas.edu/research/dunn/software.html

[2] A. K. Dunn, H. Bolay, M. A. Moskowitz, and D. A. Boas, "Dynamic imaging of cerebral blood flow using laser speckle," *J. Cereb. Blood Flow Metab.*, vol. 21, pp. 195–201, 2001.

[3] C. Ayata, A. K. Dunn, Y. Gursoy-Ozdemir, Z. Huang, D. A. Boas, and M. A. Moskowitz, "Laser speckle flowmetry for the study of cerebrovascular physiology in normal and ischemic mouse cortex," *J. Cereb. Blood Flow Metab.*, vol. 24, pp. 744–755, 2004.

[4] H. K. Shin, A. K. Dunn, P. B. Jones, D. A. Boas, M. A. Moskowitz, and C. Ayata, "Vasoconstrictive neurovascular coupling during focal ischemic depolarizations," *J. Cereb. Blood Flow Metab.*, vol. 26, pp. 1018–1030, 2006.

[5] A. J. Strong, E. L. Bezzina, P. J. B. Anderson, M. G. Boutelle, S. E. Hopwood, and A. K. Dunn, "Evaluation of laser speckle flowmetry for imaging cortical perfusion in experimental stroke studies: Quantitation of perfusion and detection of peri-infarct depolarisations," *J. Cereb. Blood Flow Metab.*, vol. 26, pp. 645–653, 2006.

[6] A. J. Strong, P. J. Anderson, H. R. Watts, D. J. Virley, A. Lloyd, E. A. Irving, T. Nagafuji, M. Ninomiya, H. Nakamura, A. K. Dunn, and R. Graf, "Peri-infarct depolarizations lead to loss of perfusion in ischaemic gyrencephalic cerebral cortex," *Brain*, vol. 130, no. 4, pp. 995–1008, 2007.

[7] T. M. Le, J. S. Paul, H. Al-Nashash, A. Tan, A. R. Luft, F. S. Sheu, and S. H. Ong, "New insights into image processing of cortical blood flow monitors using laser speckle imaging," *IEEE Trans. Med. Imag.*, vol. 26, no. 6, pp. 833–842, Jun. 2007.

[8] K. R. Forrester, C. Stewart, J. Tulip, C. Leonard, and R. C. Bray, "Comparison of laser speckle and laser doppler perfusion imaging: Measurement in human skin and rabbit articular tissue," *Med. Biol. Eng. Comput.*, vol. 40, pp. 687–697, 2002.

[9] X. W. He and J. D. Briers, "Laser speckle contrast analysis (LASCA): A real-time solution for monitoring capillary blood flow and velocity," *Proc. SPIE*, vol. 3337, pp. 98–107, 1998.

[10] J. D. Briers and X. W. He, "Laser speckle contrast analysis (LASCA) for blood flow visualization: Improved image processing," *Proc. SPIE*, vol. 3252, pp. 26–33, 1998.

[11] J. D. Briers, "Laser doppler, speckle and related techniques for blood perfusion mapping and imaging," *Physiol. Meas.*, vol. 22, pp. R35–R66, 2001.

[12] R. Bandyopadhyay, A. S. Gittings, S. S. Suh, P. K. Dixon, and D. J. Durian, "Speckle-visibility spectroscopy: A tool to study time-varying dynamics," *Rev. Sci. Instrum.*, vol. 76, pp. 093110–093110, 2005.

[13] H. Cheng and T. Q. Duong, "Simplified laser-speckle-imaging analysis method and its application to retinal blood flow imaging," *Opt. Lett.*, vol. 32, no. 15, pp. 2188–2190, Aug. 2007.

[14] S. Yuan, A. Devor, D. A. Boas, and A. K. Dunn, "Determination of optimal exposure time for imaging of blood flow changes with laser speckle contrast imaging," *Appl. Opt.*, vol. 44, no. 10, pp. 1823–1830, Apr. 2005.

[15] T. F. Chan and J. G. Lewis, "Computing standard deviations: Accuracy," *Comm. ACM*, vol. 22, no. 9, pp. 526–531, Sep. 1979.