J Real-Time Image Proc
DOI 10.1007/s11554-012-0316-z

ORIGINAL RESEARCH PAPER

Real-time processing for shape-from-focus techniques

Jawad Humayun · Aamir Saeed Malik

Received: 26 December 2011 / Accepted: 12 December 2012

Abstract An increase in the number of frames and computational complexity of the focus measure causes the shape-from-focus (SFF) method to become time consuming and occupy a lot of memory. As such, these factors become a limitation for using SFF techniques in real-time applications. However, the computational time can be significantly reduced using a parallel implementation of these methods on multiple cores. In this article, various SFF methods are compared in a parallel computing environment. The intent of this research is to analyze the speedup of various focus-measuring methods using a different number of cores to determine the optimal number of cores required for SFF applications.

Keywords Parallel computing · Shape-from-focus · Focus measure · Depth map · Speedup performance · Single program multiple data

1 Introduction

The planar projection of the real world onto a 2D image sensor causes the third dimension of the scene to be lost. A depth map is basic information required by 3D vision systems, used to approximate the 3D representation of an object or a scene. A depth map is a 2D matrix whose values correspond to the points in the real world, i.e., the points nearest to the observer and farthest from the observer, and vice versa; intermediate values correspond to real-world points that are an intermediate distance between the farthest and nearest points. In attempts to generate effective depth maps, a variety of active and passive methods are discussed in the literature [1–5]. In this paper, we consider a shape-from-focus (SFF) technique, a passive 3D shape-recovery method. Figure 1 depicts the basic image-formation phenomenon when light, after being reflected by the object, passes through the lens and falls onto an image detector (ID). From the figure, the distance between the lens and image detector is 's' whereas the distance between the lens and the focused point P' is 'v'. Thus, if we need a sharply focused image, the condition $s - v = 0$ should be met, i.e., the image detector must lie at the sharply focused point P'; otherwise, a blurred circle P'' will be formed at the image detector. This focus can be achieved by either adjusting the position of the lens 'L' or the object. If 'u' is the distance of the object from the lens, 'f' is the focal length of the lens and 's' is the distance of the lens from the image detector, their relationship can be given by the Gaussian lens law, as in Eq. (1).

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{s} \tag{1}$$

SFF is a passive optical method in which the sequence of images is acquired either by relocating the object of an unknown 3D shape in the direction of optical axes in small steps or by changing the lens position from the image sensor in the camera. The portion of the object corresponding to the 2D plane of the lens' focal plane is the sharpest and most detailed compared to the out-of-focus regions. The degree of knowledge of the focus and defocus can be calculated using focus-measuring techniques, as the depth information is provided by the

Electronic supplementary material The online version of this article (doi:10.1007/s11554-012-0316-z) contains supplementary material, which is available to authorized users.

J. Humayun · A. S. Malik (✉)
Electrical and Electronic Engineering Department, University Technology PETRONAS, Tronoh, Malaysia
e-mail: aamir_saeed@petronas.com.my

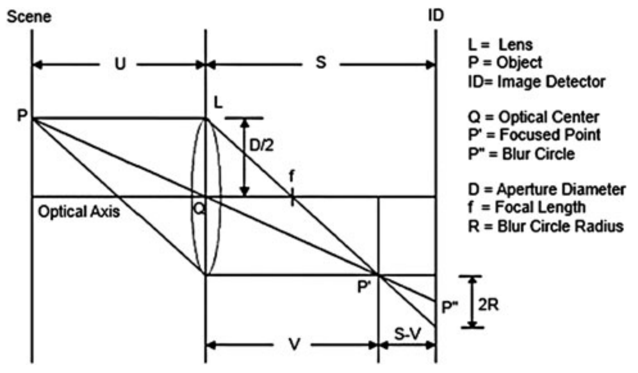


Fig. 1 Image-formation phenomenon

sharpest pixel among the sequence. For example, the construction of depth map through SFF can be performed after applying the focus measure; detailed steps of the algorithm have been described by Nayar [6]. Figure 2 shows the image sequence of a simulated cone at various focus levels. In the 15th image of the sequence, the lower base of the cone is in focus; as the image number increases, the focus plane shifts to the upper tip of the cone.

Despite being straightforward and simple, SFF methods have certain constraints that affect the depth-estimation results. For instance, as SFF methods require the full image sequence, they cannot work in a real-time dynamic environment in which the object is moving, the illumination varies from frame to frame, or the object distance varies in consecutive frames. However, one solution to this problem is a parallel implementation of SFF algorithms.

In a parallel processing environment, we can take advantage of multiple processing units to perform calculations in parallel. In this environment, complex programs can be fragmented into simpler tasks, which are then distributed to multiple processing units that run concurrently. At times, it is required to perform similar calculations on a vast amount of data; if the data chunks are distributed to multiple processing units performing similar calculations, the execution time can be significantly reduced. Likewise, frames of images in SFF can be redistributed to multiple processing units to calculate their focus measures. Hence, the purpose of testing focus-measuring techniques using different numbers of processors (P) is to analyze the speedup behavior and to then optimize the value of P for each focus-measuring algorithm.

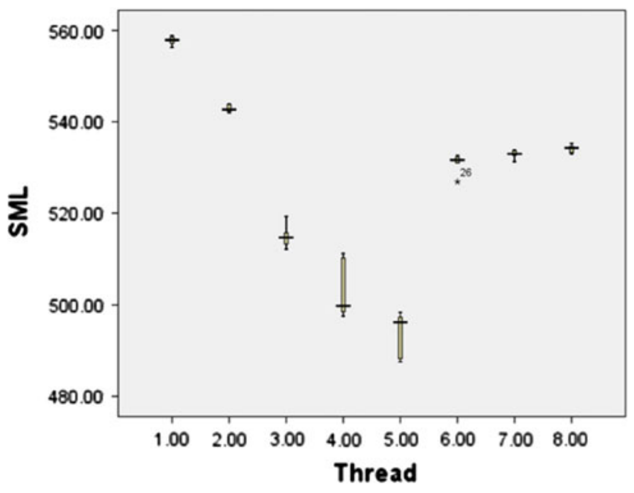


Fig. 3 *Box plot* of execution time taken by SML Gaussian interpolation algorithm for simulated cone object. The execution time is along *vertical axis* and the number of threads is along *horizontal axis*

2 Parallel computing

In parallel computing, one or more instructions of a program are concurrently executed. Larger computing problems are broken up into smaller tasks, which run simultaneously [7–9]. Parallel computing can be performed at various levels: bit-level parallelism, instruction-level parallelism, data-level parallelism, and task-level parallelism.

Bit-level and instruction-level parallelisms are carried out in the hardware of the computing machine. Bit-level parallelism can be simply defined by the word size of the processor; with an increase in the word size, the number of instructions required for the operation can be reduced if the variables are greater in length than the word size. On the other hand, for instruction-level parallelism, the instructions are reordered and regrouped in a way such that the result of the program does not change and instructions are executed in parallel. Instruction-level parallelism is performed by pipelining the instructions; pipelining methodologies vary from architecture to architecture, and include:

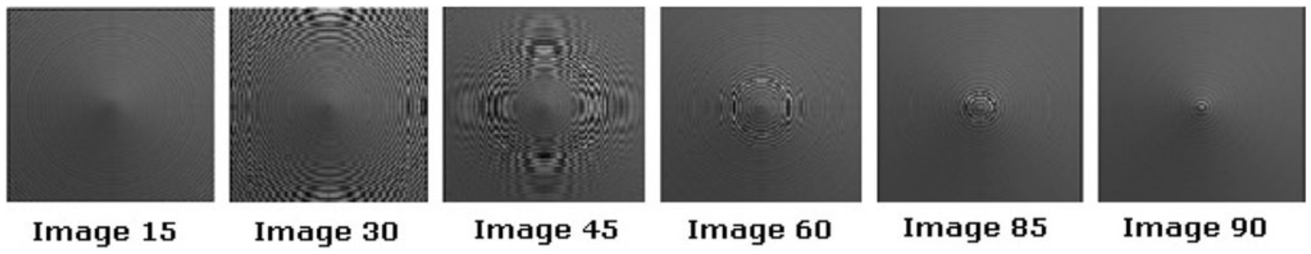


Fig. 2 SFF image sequence

five-staged pipeline, multi-scale pipelining, and superscalar architecture.

Task-level and data-level parallelism [10–16] are achieved by software applications or operating systems. In data-level parallelism (also known as loop-level parallelism), the same operations are performed over the same or different datasets. Distribution of data among the processors is then defined by conditions; e.g., if two matrices A and B are to be added, then the first-half rows of A are added to the first-half rows of B by Processor 1 and the second-half rows of A are added to the second-half rows of B by Processor 2. In this way, the execution time of the complete operation is halved. In contrast, task-level parallelism is a system in which different tasks or sets of operations are performed on the same or different datasets. Conditions are applied to assign data or parts of data to different tasks.

Programming a parallel application is difficult due to inherent dependencies [17]: flow dependency (when an instruction is dependent on the previous instruction result), anti-dependency (instruction requires value that is updated later), output dependency (when reordering causes the final result to be changed), race conditions, and constraints such as synchronization and communication between parallel tasks. To avoid potential software bugs due to these dependencies and constraints, a new set of classes (e.g., mutual exclusion lock, semaphores, etc.) have to be introduced for the parallel tasks to be synchronized while maintaining communication among them.

Different applications have different preferences according to their usage and requirements. Some applications require high-accuracy and high-resolution depth estimation, and thus, require a very large number of image sequences. In this case, the calculation of the focus measure requires more time. Hence, utilizing the multiple cores of the machine will parallelize the calculation, and in this way, time will be reduced by the speedup factor; speedup is defined in Eq. (2).

$$S_P = \frac{t_1}{t_P} \tag{2}$$

where, P is the number of processors, t_1 is execution time required by sequential algorithm, t_P is execution time required by parallel algorithm on P processors.

Ideally, the speedup factor should be linear, i.e., $S_P = P$; however, due to the execution time wasted in communication and synchronization, $S_P < P$. As such, a time comes when an increase in P does not have any effect on S_P. Thus, we have to avoid such a situation, because it causes redundant usage of resources.

The nonlinear behavior of parallel algorithms is due to the fact that not all the blocks in code can be parallelized, adding overhead to the coordination between the processors. According to Amdahl [18], as in Eq. (3), if α is the fraction of execution time required by code that cannot be parallelized, then the maximum possible speedup with parallelization of a program is:

$$S = \frac{1}{\alpha}, \tag{3}$$

i.e., if the sequential portion of a program accounts for 10 % of the runtime, we can get no more than a 10× speedup, regardless of how many processors are added. Thus, the Amdahl law assumes that the number of processors will have no effect over sequentially executable instructions.

The efficiency of an algorithm in utilizing a processor and its resources is better defined in Eq. (4):

$$E_P = \frac{S_P}{P} = \frac{t_1}{P \times t_P} \tag{4}$$

where the value typically lies between 0 and 1. For a sequential algorithm and linear speedup $E_P = 1$; thus, the efficiency provides a better understanding of parallel performance than the speedup curve.

3 Focus-measuring techniques

Many focus-measuring methods have been reported in literature [19]. In brief, focus measuring is the sharpness measuring of an image or region of an image; thus, any algorithm that measures the gradient or is used in the detection of edges can also be used for measuring the focus. The accuracy of the depth-map estimation is highly dependent on the focus-measuring algorithm as some algorithms have a higher time complexity with accuracy in sharpness measurement, and vice versa. Thus, it depends on the application requirement to determine the tradeoff between time complexity and the accuracy of results. However, the parallel implementation of complex focus-measuring algorithms can reduce the time complexity problem. In our research, we performed a parallel implementation of commonly used focus-measuring algorithms. These algorithms are described briefly in this section.

3.1 Sum-modified Laplacian (SML)

A Laplacian is obtained by summing the second derivative of an image in the x and y directions, as in Eq. (5).

$$\mathcal{L} = \frac{\partial^2 F(x,y)}{\partial x^2} + \frac{\partial^2 F(x,y)}{\partial y^2}. \tag{5}$$

As the Laplacian operator is symmetric, it is very good for detecting isolated points in an image. However, in the case of richly textured images, the horizontal and vertical components of the Laplacian operator cancel each other, generating no response. To overcome this problem, the Laplacian operator was modified here by squaring the second derivatives of the horizontal and vertical components of the Laplacian, and then, summing them, as in Eq. (6).

$$\mathrm{ML} = \left(\frac{\partial^2 F(x,y)}{\partial x^2}\right)^2 + \left(\frac{\partial^2 F(x,y)}{\partial y^2}\right)^2 \tag{6}$$

Thus, the sum of all ML pixel values in a local window is known as the sum modified Laplacian (SML) [19, 20] (Eq. 7). For richly textured images, the SML is quite effective even if applied at a single pixel; however, Nayar [6] proposed that it can also provide quite robust results when applied to weakly textured images.

$$\mathrm{SML} = \sum_{p(x,y)\in U(x_0,y_0)} \left(\frac{\partial^2 F(x,y)}{\partial x^2}\right)^2 + \left(\frac{\partial^2 F(x,y)}{\partial y^2}\right)^2. \tag{7}$$

He further proposed a Gaussian interpolation, an approximation technique that gives an even more robust and refined depth estimation. Three focus values, near the peak computed by the SML, are subsequently fitted to a Gaussian model, where the mean value is taken as the optimal depth value.

3.2 Tenenbaum (TEN)

The Tenenbaum operator [21–23] enhances or maximizes the gradient magnitude. Mathematically, it is the sum of the squared responses of the vertical and horizontal components of a Sobel mask. After applying the Tenenbaum operator, the focus measure is obtained by summing all elements in a local window, as mathematically described in Eq. (8).

$$\mathrm{TEN} = \sum_{p(x,y)\in U(x_0,y_0)} \left(G_x(x,y)^2 + G_y(x,y)^2\right)^2 \tag{8}$$

3.3 Gray-level variance (GLV)

The gray-level variance (GLV) [21–23] measures the sharpness of an image by measuring the variance of intensity values (see Eq. 9). In sharp images, this variance is higher compared to blurred images.

$$\mathrm{GLV} = \frac{1}{N-1} \sum_{p(x,y)\in U(x_0,y_0)} \left(g(x,y) - \mu_{U(x_0,y_0)}\right)^2 \tag{9}$$

where $\mu_{U(x_0.y_0)}$ is the mean of the neighboring intensity values of $U(x_0.y_0)$.

3.4 M2 focus measure

The M2 focus measure is similar to the Tenenbaum focus measure method. The M2 method is employed using a Fibonacci search, which is followed by exhaustive search. M2 is computed as in Eq. (10):

$$\mathrm{M2}(x_0,y_0) = \sum_{x=i-N}^{i+N} \sum_{y=j-N}^{j+N} \left[G_x(x,y)^2 + G_y(x,y)^2\right] \tag{10}$$

where $G_x(x,y) = G_i(x+1,y) - G_i(x,y)$ and $G_y(x,y) = G_i(x,y+1) - G_i(x,y)$

3.5 Discrete cosine transform

Few focus measures have been proposed in the discrete cosine transform (DCT) domain [24] based on the energy compaction property of its coefficients. In one such case, Baina et al. [25] proposed that the energy of the AC part of the DCT is a good approximation of the focus quality. Note that if $N \times N$ is an image block and $F(u, v)$ is its DCT, then the focus measure can be mathematically expressed as in Eq. (11).

$$F_{\mathrm{DCT}_1} = \sum_{u=1}^{N-1} \sum_{v-1}^{N-1} F(u,v)^2 \tag{11}$$

Chen [26] then suggested that the ratio between the energies of the AC and DC parts of the DCT is a better choice for the focus measure, especially for use in low contrast images. Note Eq. (12) below, where E_{AC} and E_{DC} are the respective energies of the AC and DC parts of the DCT in an image block.

$$F_{\mathrm{DCT}_2} = \frac{E_{\mathrm{AC}}}{E_{\mathrm{DC}}}. \tag{12}$$

3.6 Discrete Fourier transform (DFT)

Malik et al. [22] developed a focus-measuring algorithm based on the optical transfer function of a lens implemented in the Fourier domain, as expressed by Eq. (13).

$$\mathrm{OM}(x,y) = \sum_{x=i-N}^{i+N} \sum_{y=j-N}^{j+N} \mathrm{abs}\left[\mathcal{F}^{-1}\left(\mathcal{F}\left(|F(x,y)|^2.\left(\mathrm{e}^{-\sigma_1\left(K_x^2+K_y^2\right)} - \mathrm{e}^{-\sigma_1\left(K_x^2+K_y^2\right)}\right)\right)\right)\right] \tag{13}$$

where abs[] provides only the real part of the complex number, $\mathcal{F}$ is the Fourier transform, and $\mathcal{F}^{-1}$ is the inverse Fourier transform. In addition, $F(x, y)$ is the image frame, and the expression $\left(\mathrm{e}^{-\sigma_1\left(K_x^2+K_y^2\right)} - \mathrm{e}^{-\sigma_1\left(K_x^2+K_y^2\right)}\right)$ represents a band pass filter with cutoff frequencies $\sigma_1\left(K_x^2 + K_y^2\right)$ and $\sigma_2\left(K_x^2 + K_y^2\right)$.

3.7 Discrete wavelet transform

The discrete wavelet transform (DWT) [27] is applied to the local window of an image frame in order to obtain the approximate (low-frequency component) and details (high-frequency component) of the image. The ratio of the high-frequency component to the low-frequency component gives the focus measure.

3.8 Histogram entropy (HE)

In the histogram of a sharply focused image, two spikes are usually observed; each corresponds to one side of an edge. Blurred images do not exhibit this property, though it can be measured through histogram entropy (HE) using the mathematical relation in Eq. (14).

$$\mathrm{HE} = -\sum_i P(I) \cdot \ln(P(I)), \quad P(I) \neq 0 \tag{14}$$

where I is the intensity level and $P(I)$ is the frequency of occurrence of I in an image. HE is a minimum when $P(I)$ is zero for all except one value of I, and it is a maximum when all $P(I)$ are equal. Therefore, the sharp edge will have less entropy compared to the blurred edge.

4 Setup and implementation

This section describes the parallel implementation of SFF algorithms. Here, the sequential algorithm is modified to execute in parallel; only the focus-measuring part of the SFF algorithm has been parallelized by equally distributing the number of frames of test samples to the CPU cores. In this way, each core contributes in parallel to the calculation of the focus measure, e.g., if a test sample has 80 frames, then on a two-core machine, the first 40 frames (1–40) will be assigned to core 1 and the next 40 frames (41–80) will be assigned to core 2. The CPU time is noted only for instructions for calculating the focus measure, whereas the time depleted for the remainder of the SFF algorithm is not counted (as it is same in every algorithm). Moreover, for each focus-measuring algorithm and each number of cores, speedup is calculated by repeating the experiment five times.

Five different samples are used for the 3D shape recovery. For simplified and speedy results, all samples were 40 × 40 center cropped; details of the test samples are given in Table 1. A machine having 2× (2.26 GHz Quad Core Intel Xeon) processors was used to run the code. The machine had 6 GB DDR3 RAM running MAC OS X ver 10.6.8.

5 Results and discussion

Tables 2, 3, 4, 5 and 6 show the speedup for each of the test objects. Figures 4, 5, 6, 7, 8, 9, 10 and 11 subsequently depict the speedup versus number of threads behavior/plot for SML, TEN, M2, GLV, DCT, DFT, HE, and DWT, respectively. Moreover, each figure shows the plot for all five samples. Table 2 also shows the upper- and lower-bound confidence interval of 95 % for speedup. Figure 3 is the boxplot of execution time taken by SML Gaussian interpolation algorithm for simulated cone object. It can be observed that very few outliers are present, i.e., variation of execution time by the same algorithm for the same object with the same number of threads is minute. All the other cases show similar behavior.

When analyzing the behavior of the SML using a Gaussian interpolation in parallel computing (Fig. 4), it can be seen that the speedup increases for all objects with the use of up to three cores; after which, it starts declining. Here, 1.31 is the maximum speedup achieved, at an efficiency of 43 %. Surprisingly, all objects generated a diverse speedup plot. However, due to the use of Gaussian interpolations, the time of search for local maxima depends on the data, i.e., for some images, local maxima are found early, whereas for others they are found late.

Based on the analysis of the DCT speedup (Fig. 8), it can be concluded that a higher number of object images improves the efficiency. Simulated and real cones have the same and highest number of images (Table 1) and their speedup curves are also seen to be both identical and the

Table 1 Test samples

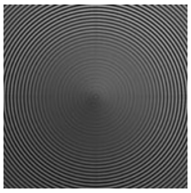
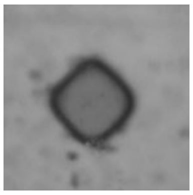
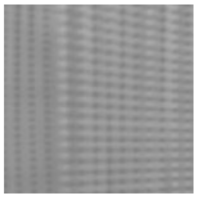
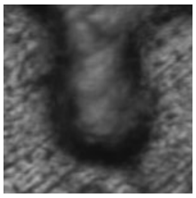
Test object	Simulated cone	Real cone	TFT LCD cell	Slanted plane	US 1 cent
Number of frames	97	97	60	87	68
Single frame					

Table 2 Results for simulated cone

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
SML with Gaussian interpolation								
	557.91	542.23	513.24	510.26	498.36	526.87	532.65	534.27
	558.62	543.87	515.68	498.56	497.33	530.99	533.78	535.27
	557.21	541.95	514.63	497.57	488.25	531.62	531.27	534.62
	556.27	542.68	519.27	511.2	496.22	532.66	532.98	533.29
	558.92	543.81	512.12	499.8	487.56	532.3	533.65	532.96
Average	557.79	542.91	514.99	503.48	493.54	530.89	532.87	534.08
Execution time confidence interval of 95 % (upper range)	559.12	544.01	518.40	511.76	500.01	533.78	534.11	535.26
Execution time confidence interval of 95 % (lower range)	556.45	541.80	511.57	495.18	487.07	527.98	531.61	532.9
Speedup	1	1.02	1.08	1.1	1.13	1.05	1.04	1.04
Speedup confidence interval of 95 % (upper range)	1	1.027	1.07	1.09	1.11	1.04	1.046	1.044
Speedup confidence Interval of 95 % (Lower Range)	1	1.02	1.08	1.12	1.14	1.05	1.04	1.04
Tenenbaum (TEN)								
	245.39	142.3	105.94	89.51	84.1	85.72	83.23	85.81
	247.05	141.1	105.97	90.52	85.36	85.32	82.35	83.29
	246.69	141.47	105.49	90.7	84.39	85.48	81.78	80.6
	246.44	141.32	106.7	91.19	84.99	85.32	82.67	83.56
	249.01	140.05	106.93	90.68	85.16	84.49	83.23	84.18
Average	246.92	141.25	106.21	90.52	84.8	85.27	82.65	83.49
Execution time confidence interval of 95 % (upper range)	248.56	142.25	106.94	91.28	85.46	85.84	83.41	85.83
Execution time confidence interval of 95 % (lower range)	245.27	140.24	105.47	89.75	84.13	84.69	81.88	81.14
Speedup	1	1.74	2.35	2.72	2.91	2.89	2.98	2.95
Speedup confidence interval of 95 % (upper range)	1	1.74	2.32	2.72	2.90	2.89	2.97	2.89
Speedup confidence interval of 95 % (lower range)	1	1.74	2.32	2.73	2.91	2.89	2.99	3.02
M2								
	88.82	55.79	45.62	39.6	37.5	37.04	36.66	37.22
	88.35	54.66	43.7	39.56	38.11	37.27	36.12	36.72
	88.99	55.13	43.45	38.57	38.6	42.33	36.82	36.98
	89.02	55.75	43.45	38.88	38.09	36.48	37.13	36.91
	90.52	55.19	43.15	39.17	37.91	36.89	37.16	37.48
Average	89.14	55.3	43.87	39.16	38.04	38	36.78	37.06
Execution time confidence interval of 95 % (upper range)	90.15	55.89	45.11	39.70	38.53	41.02	37.30	37.42
Execution time confidence interval of 95 % (lower range)	88.12	54.71	42.63	38.60	37.54	34.97	36.25	36.69
Speedup	1	1.61	2.03	2.27	2.34	2.34	2.42	2.4
Speedup confidence interval of 95 % (upper range)	1	1.61	1.99	2.27	2.33	2.19	2.41	2.40
Speedup confidence interval of 95 % (Lower Range)	1	1.61	2.06	2.28	2.34	2.51	2.43	2.40
Grey level variance (GLV)								
	22.12	10.92	7.9	5.84	4.9	4.22	3.94	3.49
	21.49	10.91	7.84	5.83	4.85	4.16	3.82	3.58
	21.45	10.91	7.8	5.84	4.83	4.2	3.81	3.52
	21.41	10.95	7.8	5.85	4.85	4.2	3.81	3.55
	21.42	10.91	7.814	5.83	4.84	4.22	3.82	3.56
Average	21.58	10.92	7.83	5.84	4.85	4.2	3.84	3.54

Table 2 continued

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
Execution time confidence interval of 95 % (upper range)	21.956	10.942	7.883	5.848	4.887	4.23	3.91	3.584
Execution time confidence interval of 95 % (lower range)	21.2	10.898	7.779	5.828	4.82	4.17	3.77	3.496
Speedup	1	1.97	2.75	3.69	4.44	5.13	5.61	6.09
Speedup confidence interval of 95 % (upper range)	1	2.00	2.78	3.75	4.49	5.19	5.61	6.12
Speedup confidence interval of 95 % (lower range)	1	1.94	2.72	3.63	4.39	5.08	5.62	6.06
Discrete cosine transform (DCT)								
	27.24	19.95	15.47	13.02	11.59	10.69	10.34	10.57
	27.12	20.26	15.45	13.19	11.72	10.64	10.43	10.19
	26.88	20.23	16.01	12.92	11.6	10.66	10.3	10.73
	26.96	20.33	15.57	12.96	11.69	10.73	10.48	10.37
	26.87	20.02	15.53	12.71	11.53	10.65	10.84	10.45
Average	27.02	20.16	15.61	12.96	11.63	10.67	10.48	10.46
Execution time confidence interval of 95 % (upper range)	27.214	20.362	15.893	13.176	11.722	10.719	10.744	10.715
Execution time confidence interval of 95 % (lower range)	26.814	19.954	15.319	12.744	11.53	10.629	10.212	10.209
Speedup	1	1.35	1.74	2.1	2.34	2.55	2.59	2.6
Speedup confidence interval of 95 % (upper range)	1	1.33	1.71	2.06	2.32	2.53	2.53	2.53
Speedup confidence interval of 95 % (lower range)	1	1.34	1.75	2.10	2.32	2.52	2.62	2.62
Discrete Fourier transform (DFT)								
	155.29	88.13	60.93	54.22	51.41	49.24	50.19	50.78
	156.59	89.77	64.73	55.18	50.39	49.95	51.96	50.92
	157.19	90.27	69.19	55.15	51.33	49.89	51.86	51.1
	156.89	90.6	68.16	55.49	52.08	49.67	51.61	51.01
	156.38	88.3	68.23	55.31	50.58	50.13	51.28	50.86
Average	156.47	89.42	66.24	55.07	51.16	49.78	51.38	50.94
Execution time confidence interval of 95 % (upper range)	157.37	90.824	70.497	55.684	52.006	50.2	52.268	51.089
Execution time confidence interval of 95 % (lower range)	155.566	88.004	62	54.457	50.31	49.352	50.492	50.779
Speedup	1	1.75	2.36	2.84	3.06	3.14	3.05	3.07
Speedup confidence interval of 95 % (upper range)	1	1.73	2.23	2.82	3.02	3.13	3.01	3.08
Speedup confidence interval of 95 % (lower range)	1	1.76	2.50	2.85	3.09	3.15	3.08	3.06
Histogram entropy (HE)								
	90.13	63.78	54.43	51.36	50.66	49.81	48.74	48.83
	89.67	63.02	53.84	50.47	49.91	48.65	48.71	48.62
	90.12	63.25	53.88	49.6	49.53	49.11	48.32	48.35
	89.95	63.47	53.96	50.12	49.57	48.93	48.06	48.51
	89.65	63.01	54.03	50.52	49.94	48.59	48.37	48.58
Average	89.9	63.3	54.03	50.41	49.92	49.02	48.44	48.58
Execution time confidence interval of 95 % (upper range)	90.194	63.71	54.321	51.213	50.485	49.627	48.795	48.795
Execution time confidence interval of 95 % (lower range)	89.613	62.901	53.735	49.615	49.359	48.409	48.085	48.361
Speedup	1	1.42	1.66	1.78	1.8	1.83	1.85	1.85
Speedup confidence interval of 95 % (upper range)	1	1.41	1.66	1.76	1.78	1.81	1.84	1.84
Speedup confidence interval of 95 % (lower range)	1	1.42	1.66	1.80	1.81	1.85	1.86	1.85

Table 2 continued

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
Discrete wavelet transform (DWT)								
	62.42	34.8	25.66	20.57	19.24	18.74	17.5	17.62
	62.4	34.93	24.96	20.73	19.92	18.4	17.21	17.88
	62.46	34.18	25.12	20.17	19.38	18.61	17.18	17.93
	62.47	33.87	25.12	20.64	19.62	18.72	17.27	17.71
	62.7	34.38	24.96	20.45	19.34	18.49	17.36	17.89
Average	62.49	34.43	25.16	20.51	19.5	18.59	17.3	17.81
Execution time confidence interval of 95 % (upper range)	62.64	34.975	25.522	20.781	19.839	18.774	17.465	17.972
Execution time confidence interval of 95 % (lower range)	62.334	33.889	24.806	20.243	19.161	18.41	17.143	17.64
Speedup	1	1.81	2.483	3.046	3.204	3.361	3.611	3.509
Speedup confidence interval of 95 % (upper range)	1	1.79	2.45	3.01	3.15	3.33	3.58	3.48
Speedup confidence interval of 95 % (lower range)	1	1.83	2.51	3.07	3.25	3.38	3.63	3.53

Table 3 Results for real cone

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
SML (GI)	1	1.16	1.20	1.22	1.18	1.24	1.24	1.22
TEN	1	1.71	2.28	2.70	2.822	2.88	2.93	2.82
M2	1	1.58	1.94	2.20	2.203	2.26	2.28	2.22
GLV	1	1.90	2.74	3.59	4.42	5.028	5.60	6.01
DCT	1	1.31	1.70	2.09	2.29	2.52	2.55	2.57
DFT	1	1.76	2.52	2.80	3.03	3.04	2.97	2.97
HE	1	1.42	1.64	1.77	1.76	1.77	1.79	1.77
DWT	1	1.78	2.52	3.02	3.17	3.39	3.50	3.61

Table 4 Results for TFT LCD

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
SML (GI)	1	1.21	1.31	1.29	1.28	1.27	1.26	1.23
TEN	1	1.74	2.33	2.80	2.92	2.99	2.91	2.82
M2	1	1.60	2.01	2.27	2.32	2.34	2.30	2.23
GLV	1	1.91	2.70	3.42	4.09	4.99	5.35	5.67
DCT	1	1.13	1.42	1.53	1.68	1.80	1.90	1.83
DFT	1	1.76	2.53	2.96	3.02	3.09	3.05	3.00
HE	1	1.42	1.66	1.78	1.78	1.80	1.80	1.80
DWT	1	1.77	2.45	2.99	3.204	3.373	3.431	3.409

most efficient. In contrast, a TFT LCD has the least amount of images and its speedup curve is the least efficient. A radical increase in speedup can be seen for up to five workers for the DCT.

Unlike the DCT and SML with Gaussian interpolations, all other methods for focus measure, i.e., TEN (Fig. 4), M2 (Fig. 6), GLV (Fig. 7), DFT (Fig. 9), HE (Fig. 10), and DWT (Fig. 11), have similar speedup characteristics with respect to data or specimens under observation. However, the speedup efficiency varies from one focus measure to another. Among these algorithms, GLV (Fig. 7) has the best speedup performance, which is almost linear up to four workers. In fact, GLV is seen to be vastly superior to all other algorithms in terms of speedup performance. Moreover, except for DCT and SML with Gaussian interpolations, all other algorithms have speedup curves that shoot

Table 5 Results for slanted plane

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
SML (GI)	1	1.09	1.13	1.15	1.14	1.10	1.12	1.10
TEN	1	1.73	2.31	2.76	2.80	2.87	2.89	2.90
M2	1	1.59	1.97	2.22	2.23	2.26	2.27	2.24
GLV	1	1.93	2.77	3.54	4.37	5.12	5.52	5.89
DCT	1	1.30	1.66	2.02	2.20	2.39	2.43	2.45
DFT	1	1.77	2.53	2.94	2.96	3.04	2.98	2.98
HE	1	1.41	1.64	1.76	1.75	1.78	1.78	1.78
DWT	1	1.80	2.48	3.10	3.24	3.377	3.50	3.611

Table 6 Results for microscopic coin

	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
SML (GI)	1	1.10	1.15	1.15	1.15	1.15	1.13	1.12
TEN	1	1.74	2.28	2.80	2.85	2.86	2.92	2.88
M2	1	1.62	2.00	2.30	2.30	2.30	2.34	2.31
GLV	1	1.94	2.78	3.48	4.31	4.86	5.61	5.83
DCT	1	1.19	1.63	1.64	1.95	2.06	2.08	2.05
DFT	1	1.76	2.56	2.95	3.04	3.09	3.09	3.05
HE	1	1.41	1.66	1.79	1.77	1.79	1.81	1.80
DWT	1	1.79	2.456	2.99	3.18	3.29	3.48	3.42

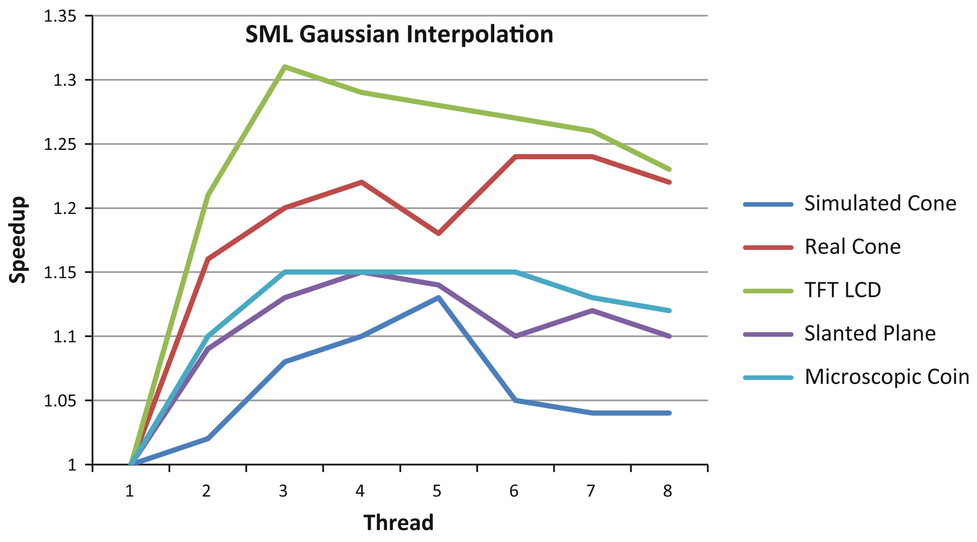


Fig. 4 Speedup chart for SML Gaussian interpolation

upwards until they reach thread 4 (Figs. 5, 6, 9, 10, 11), after which they start losing speedup efficiency or become nearly constant. Thus, it can be concluded that the optimal number of processors or workers for improving the speedup efficiency of most focus-measuring algorithms is four.

Figures 12 and 13 show the combined speedup results for various focus-measuring techniques applied over simulated cones and TFT LCD cells. These figures provide a more focused analysis of speedup among the different algorithms; both depict the common speedup characteristics. The GLV focus measure apparently shows the best speedup performance, whereas the SML with Gaussian interpolation is least efficient in terms of parallel implementation. With respect to the other algorithms, a similar trend follows for both specimens, i.e., speedup

Fig. 5 Speedup chart for TEN

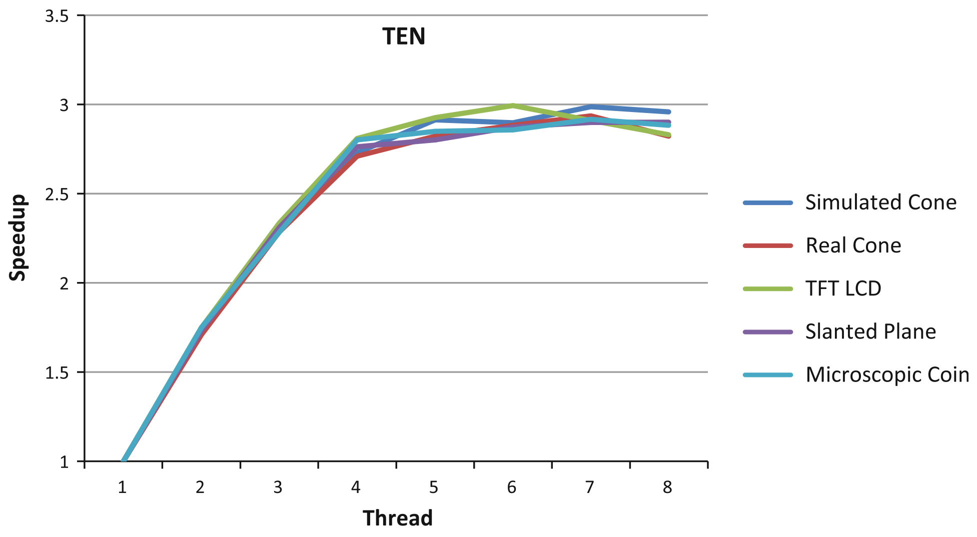


Fig. 6 Speedup chart for M2

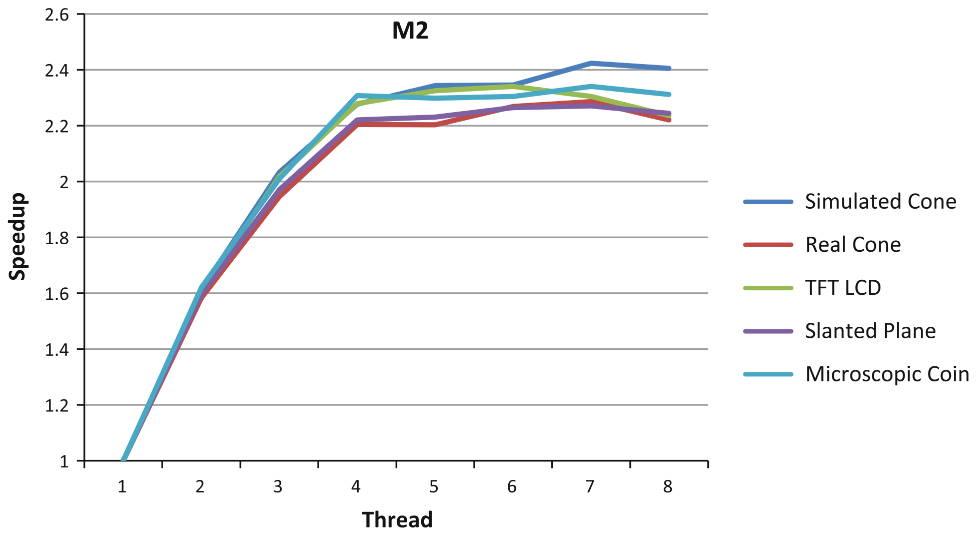


Fig. 7 Speedup chart for GLV

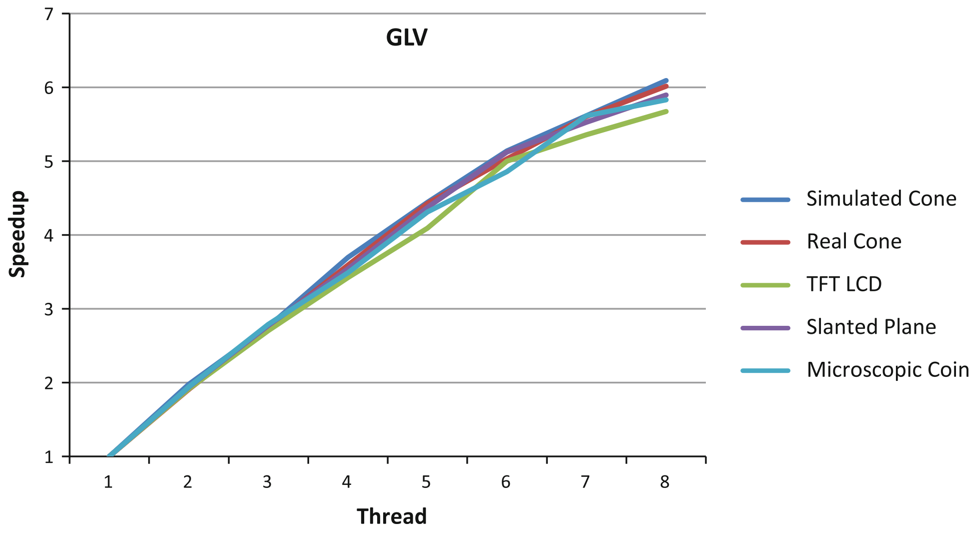


Fig. 8 Speedup chart for DCT

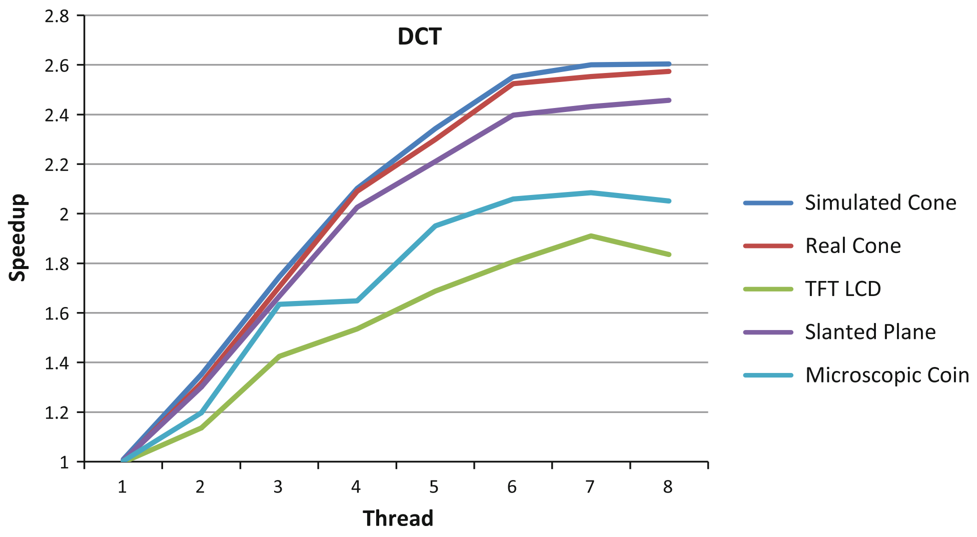


Fig. 9 Speedup chart for DFT

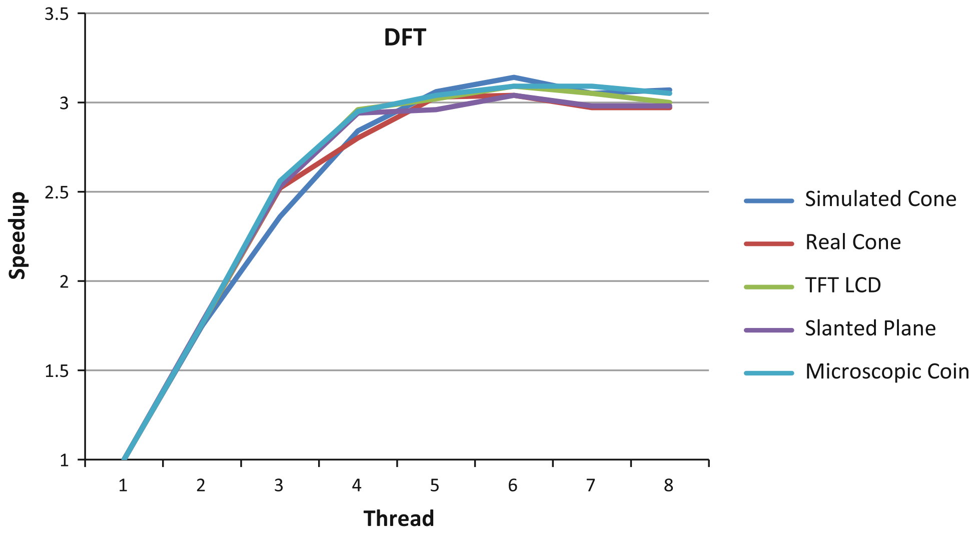
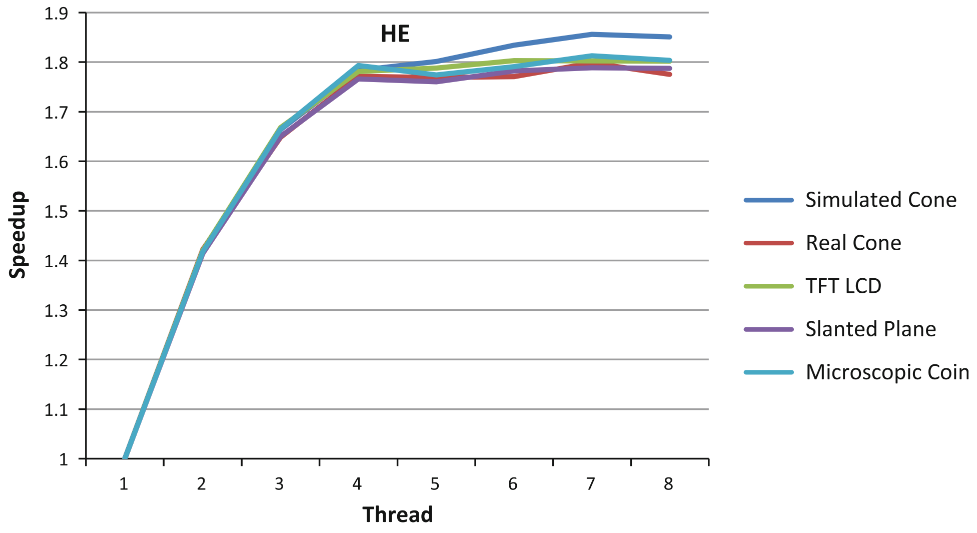


Fig. 10 Speedup chart for HE



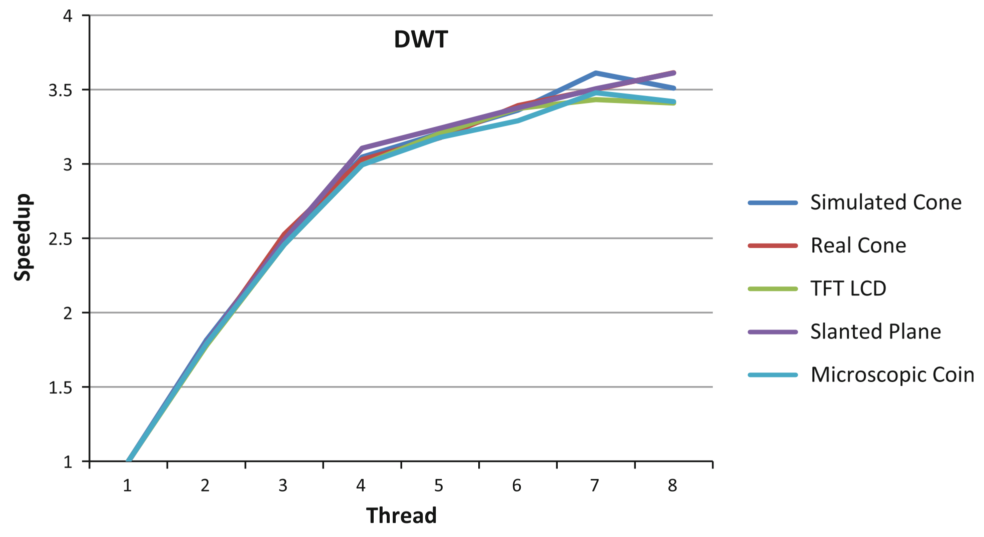


Fig. 11 Speedup chart for DWT

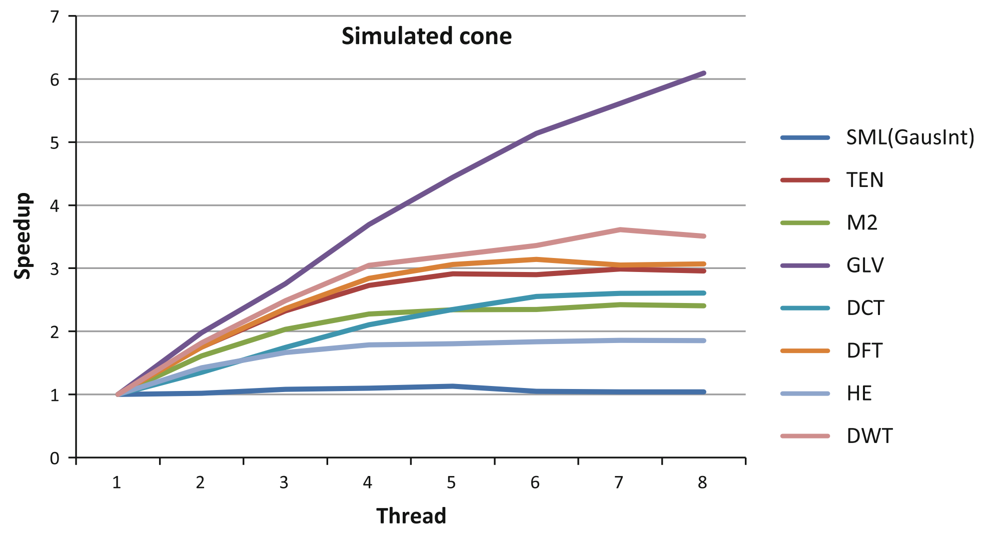


Fig. 12 Speedup results of simulated cone at various focus measures

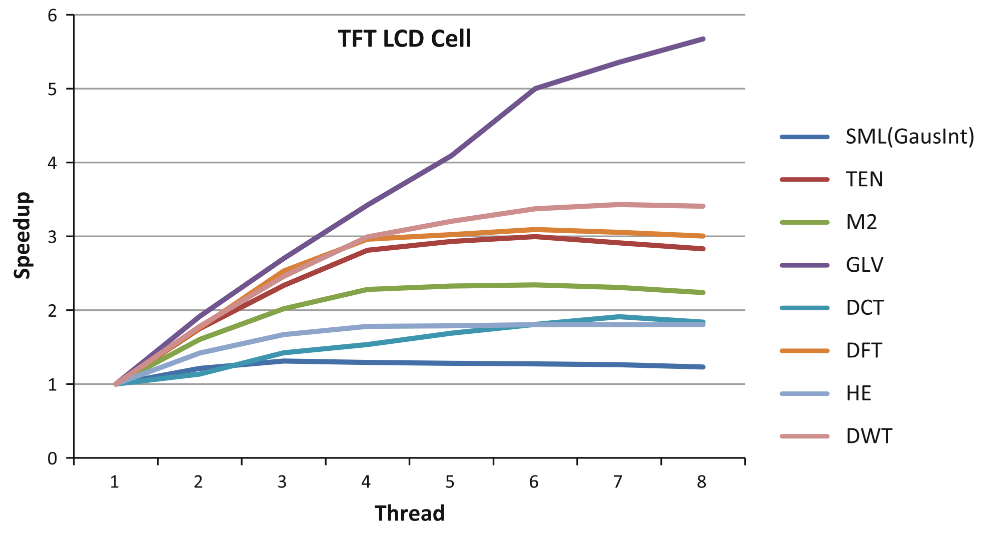


Fig. 13 Speedup results of TFT LCD cell at various focus measures

performance decreases from DWT, DFT, TEN, M2, to HE, whereas the performance curve of DCT and M2 compete with each other in various cases.

6 Conclusions

Parallel processing compensates for the need for high frequency clock cycles in hardware, which lowers the power consumption cost and uses of additional hardware. On the other hand, by introducing parallel processing in the software, optimal use can be made of the hardware resources in the machine. It was also noted that parallel processing is more complex than the sequential processing and may generate inconsistent results if not carefully programmed; however, such inconsistencies can be avoided by use of special classes that effectively allocate resources while the communication and synchronization of different processes run in parallel. As a continuing effort, since some complex tasks cannot be fully parallelized, the sequential part of these processes can cause the speedup curve to become constant or decline—a condition that requires further investigation.

In this article, we parallelized a few of the focus-measuring algorithms known in the literature. These algorithms have been parallelized at the data/loop-level (Single Program Multiple Data) [12, 15]. We then worked to determine the optimal number of workers required for parallelizing the focus-measuring algorithms. During this analysis, some interesting facts were unveiled. First, the speedup efficiency of some focus-measuring algorithms was found to depend on the data content and quantity. In addition, higher searches for local maxima in the Gaussian interpolation of SML cause a decrease in the speedup efficiency, and the speedup efficiency of the DCT focus measure increases with an increase in the number of images in the sample. The GLV focus measure has excellent speedup performance as its speedup curve is approximately linear. Importantly, for most of the algorithms discussed in this article, the optimal number of processors required for parallelization is four; this optimized value is relative to the machine used in experimentation. Thus, the parallelization of focus-measuring algorithms is found to be advantageous for SFF, i.e., applications that require less time or use large image sequences for accuracy and robustness can make use of parallelization to expedite their processing.

References

1. Yemez, Y., Schmitt, F.: 3D reconstruction of real objects with high resolution shape and texture. Image Vis. Comput. **22**, 1137–1153 (2004)
2. Dipanda, A., Woo, S.: Towards a real-time 3D shape reconstruction using a structured light system. Pattern Recognit. **38**, 1632–1650 (2005)
3. Mamassian, P., Knill, D.C., Kersten, D.: The perception of cast shadows. Trends Cogn. Sci. **2**, 288–295 (1998)
4. Norman, J.F., Lee, Y.L., Phillips, F., Norman, H.F., Jennings, L.R., McBride, T.R.: The perception of 3-D shape from shadows cast onto curved surfaces. Acta Psychol. **131**, 1–11 (2009)
5. Li, M., Kambhamettu, C., Stone, M.: Nonrigid motion recovery for 3D surfaces. Image Vis. Comput. **25**, 250–261 (2007)
6. Nayar, S.K., Nakagawa, Y.: Shape from focus: an effective approach for rough surfaces. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 218–225 (1990)
7. Liu, F., Sosonkina, M.: A multilevel parallelism support for multi-physics coupling. Proc Comput Sci **4**, 261–270 (2011)
8. Plaza, A.: Parallel techniques for information extraction from hyperspectral imagery using heterogeneous networks of workstations. J. Parallel Distrib. Comput. **68**, 93–111 (2008)
9. Sutter, B.: On the use of subword parallelism in medical image processing. Parallel Comput. **24**, 1537–1556 (1998)
10. Beynon, M.: Processing large-scale multi-dimensional data in parallel and distributed environments. Parallel Comput. **28**, 827–859 (2002)
11. Chang, W., Ho, M.S.: Exploitation of parallelism to nested loops with dependence cycles. J. Syst. Archit. **50**, 729–742 (2004)
12. Huang, T.: A practical run-time technique for exploiting loop-level parallelism. J. Syst. Softw. **54**, 259–271 (2000)
13. Romero, L., Ortigosa, E., Zapata, E.: Data-task parallelism for the VMEC program. Parallel Comput. **27**, 1347–1364 (2001)
14. Nicolescu, C.: A data and task parallel image processing environment. Parallel Comput. **28**, 945–965 (2002)
15. Singh, A.: An integrated performance analysis tool for SPMD data-parallel programs. Parallel Comput. **23**, 1089–1112 (1997)
16. Hermenegildo, M.: Relating data-parallelism and parallelism in logic programs. Comput. Lang. **22**, 143–163 (1996)
17. Hossain, M.: Impact of data dependencies in real-time high performance computing. Microprocess. Microsyst. **26**, 253–261 (2002)
18. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. Business **30**, 19–20 (2007)
19. Aslantas, V., Kurban, R.: A comparison of criterion functions for fusion of multi-focus noisy images. Optics Commun. **282**, 3231–3242 (2009)
20. Zhao, H., Li, Q., Feng, H.: Multi-focus color image fusion in the HSI space using the sum-modified-Laplacian and a coarse edge map. Image Vis. Comput. **26**, 1285–1295 (2008)
21. Rahmat, R., Malik, A.S., Faye, I., Kamel, N.S.: An overview of lulu operators and discrete pulse transform for image analysis. Imag. Sci. J. **59**(5) (2011)
22. Rahmat, R., Malik, A.S., Kamel, N.S.: 3-D content generation using optical passive reflective techniques. The 15th IEEE International Symposium on Consumer Electronics, Singapore (2011)
23. Rahmat, R., Malik, A.S., Kamel, N.S.: Comparison of LULU and median filter for image denoising. 3rd Conference on Signal Acquisition and Processing (ICSAP2011), Singapore (2011)
24. Haghighat, M.B.A., Aghagolzadeh, A., Seyedarabi, H.: Multi-focus image fusion for visual sensor networks in DCT domain. Comput. Electr. Eng. (2011)
25. Baina, J., Dublet, J.: Automatic focus and iris control for video cameras. Image Processing, IEEE Fifth International Conference on Image Processing and its Applications, pp. 232–235 (1995)
26. Chen, H.H.: Robust focus measure for low-contrast images. Digest of Technical Papers International Conference on Consumer Electronics, IEEE, pp. 69–70 (2006)

27. Kautsky, J.: A new wavelet-based measure of image focus. Pattern Recognit. Lett. **23**, 1785–1794 (2002)

Author Biographies

Jawad Humayun is a researcher in Department of Electrical and Electronic Engineering at Universiti Teknologi PETRONAS. He received his B.Sc. in EE from National University of Computer and Emerging Sciences (NUCES), Lahore, Pakistan. His research interests include Medical imaging, 3D imaging and Machine Learning.

Aamir Saeed Malik is an Associate Professor in Department of Electrical and Electronic Engineering and member of Centre for Intelligent Signal and Imaging Research at Universiti Teknologi PETRONAS. He received his BS in EE from Pakistan while MS and Ph.D. from Republic of Korea. He has more than 8 years of working experience. His research interests include 3D imaging, medical imaging and CBIR.