SCIENCE CHINA
Technological Sciences

Special Topic: Digital Image Correlation
•Article•

January 2018 Vol.61 No.1: 74–85
https://doi.org/10.1007/s11431-017-9168-0

Heterogeneous parallel computing accelerated iterative subpixel digital image correlation

HUANG JianWen[1,2], ZHANG LingQi[2], JIANG ZhenYu[1,3*], DONG ShouBin[2*], CHEN Wei[1], LIU YiPing[1], LIU ZeJia[1], ZHOU LiCheng[1] & TANG LiQun[1]

[1] *State Key Laboratory of Subtropical Building Science, School of Civil Engineering and Transportation, South China University of Technology, Guangzhou 510640, China;*
[2] *School of Computer Science, South China University of Technology, Guangzhou 510640, China;*
[3] *The State Key Laboratory of Nonlinear Mechanics, Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China*

Received July 12, 2017; accepted October 30, 2017; published online December 15, 2017

Parallel computing techniques have been introduced into digital image correlation (DIC) in recent years and leads to a surge in computation speed. The graphics processing unit (GPU)-based parallel computing demonstrated a surprising effect on accelerating the iterative subpixel DIC, compared with CPU-based parallel computing. In this paper, the performances of the two kinds of parallel computing techniques are compared for the previously proposed path-independent DIC method, in which the initial guess for the inverse compositional Gauss-Newton (IC-GN) algorithm at each point of interest (POI) is estimated through the fast Fourier transform-based cross-correlation (FFT-CC) algorithm. Based on the performance evaluation, a heterogeneous parallel computing (HPC) model is proposed with hybrid mode of parallelisms in order to combine the computing power of GPU and multicore CPU. A scheme of trial computation test is developed to optimize the configuration of the HPC model on a specific computer. The proposed HPC model shows excellent performance on a middle-end desktop computer for real-time subpixel DIC with high resolution of more than 10000 POIs per frame.

digital image correlation (DIC), inverse compositional Gauss-Newton (IC-GN) algorithm, heterogeneous parallel computing, graphics processing unit (GPU), multicore CPU, real-time DIC

Citation: Huang J W, Zhang L Q, Jiang Z Y, et al. Heterogeneous parallel computing accelerated iterative subpixel digital image correlation. Sci China Tech Sci, 2018, 61: 74–85, https://doi.org/10.1007/s11431-017-9168-0

1 Introduction

Iterative algorithms have occupied a dominant position in subpixel digital image correlation (DIC) thanks to their superior accuracy (i.e. lower systematic error and random error) over the non-iterative algorithms [1,2]. However, these iterative algorithms suffer from the low computation efficiency (one order of magnitude lower than their non-iterative counterpart) as the iterative procedure is very time consuming. The demanding of high speed subpixel DIC without loss of accuracy becomes urgent in recent years due to the rapid advance of digital cameras with high speed and high resolution, which leads to the proliferation of images as well as the pixels in each image to be processed. Moreover, the intensive study on the deformation process (particularly the crack propagation during fracture or fatigue tests) also creates a growing demand for real-time DIC [3,4].

In the past decade, various approaches have been developed to speed up the iterative subpixel DIC. Among the effort devoted to optimizing the DIC algorithm, the introduction of the inverse compositional Gauss-Newton (IC-

* Corresponding authors (email: zhenyujiang@scut.edu.cn; sbdong@scut.edu.cn)

GN) algorithm into DIC could be considered as a milestone [5,6]. The IC-GN algorithm [7], mathematically equivalent to the classic forward additive Newton-Raphson (FA-NR) algorithm [8], uses a smart alignment strategy during the iterative procedure, which makes it 3–5 times faster than the FA-NR algorithm [6], while keeping the high accuracy and resistance to noise [9–11]. In addition, the application of the precomputed global look-up table of interpolation coefficients [12,13] eliminates the repetitive calculation during the construction of the warped target subset in iterations and leads to a speed-up of another several times. Nevertheless, the iterative subpixel DIC with the aforementioned improvements achieves a speed of only thousands of points of interest per second (POI/s) [14,15], which is still far below the expectation.

Further exploitation of computing hardware, i.e. parallel computing technology, sheds light on the solution to high speed iterative subpixel DIC [16,17]. The multicore CPUs (central processing units) and GPU (graphics processing unit) devices equipped in current computers demonstrated unprecedented power in boosting the computation efficiency of iterative DIC for even orders of magnitude. The multicore CPU parallel computing has been introduced into iterative DIC by Correlated Solutions Inc since 2009 (http://www.correlatedsolutions.com/installs/Vic-2D-2009-Manual.pdf). Recently, Shao et al. [9] and Pan et al. [10] reported their study almost simultaneously about the acceleration of IC-GN algorithm-based DIC using multicore CPU. In their implementation, multiple POIs are processed in parallel by CPU threads. It is noteworthy that the initial guess transfer scheme is used by both groups for the IC-GN algorithm, which may limit the parallelism. But actually the number POIs fed into the queue to be processed increases geometrically with the number of processed POIs, which makes the processing reach the maximum parallel degree of a multicore CPU in a very short time. A dramatic speed-up of about seven times (30414–43667 POI/s, using a 21×21-pixel subset) was achieved by using eight parallel threads on a quad-core CPU, compared with the implementation using a single CPU thread.

The application of GPUs has been far beyond their originally designed purpose for accelerating the visualization of 3D sceneries. The developments of computing platform and programming model for GPU parallel computing, in particular the compute unified device architecture (CUDA) released by NVIDIA (https://en.wikipedia.org/wiki/CUDA), stimulate the extensive use of GPUs in a wide variety of areas [18,19]. Acceleration of DIC using GPU parallel computing could be traced back to 2009 when Leclerc et al. [20] introduced GPU into the mesh procedure for finite element-based DIC. The implementation reduced the computation time for a 1024×1024-pixel image pair down to seven hundred seconds. Leclerc et al. [21] further extended the application of GPU parallel computing to their voxel-based digital volume correlation (DVC) method, in which GPU is used to solve the linear system including millions of degrees of freedom.

In comparison with the global DIC and DVC methods based on finite elements, local DIC and DVC methods based on subsets (or sub-volumes) seem to be more suitable for parallel computing technology due to their nature of windowed processing. GPU parallel computing can be simply used in cross-correlation-based integer-pixel DIC algorithms for magnetic resonance image processing [22], crack propagation analysis [23] and deformation measurement of human body (wrist and calf) [24]. The speed-up ratio of the GPU version to the serial CPU version attained in refs. [21–24] varies from 10 to 23.

The GPU parallel computing shows the surprising capability to speed up the iterative subpixel DIC. Zhang et al. [25] proposed a GPU-accelerated path-independent DIC (PiDIC) method, which estimates the initial guess for the IC-GN algorithm at each POI independently using the fast Fourier transform-based cross-correlation (FFT-CC) algorithm [15]. The implementation achieved a computation speed of 1.6×10^5 POI/s (33×33-pixel subset) without sacrifice of high accuracy using a low-end graphics card (NVIDIA GTX 760). This high computation efficiency makes the real-time subpixel DIC feasible at a video recording rate (e.g. 30 fps). Very recently, an extremely high record of computation speed (2.5×10^6 POI/s) was created by Le Besnerais et al. [26], who implemented an iterative Lucas-Kanade algorithm-based DIC method on a high-performance GPU board (NVIDIA Titan). This success could be attributed to two factors: (i) zero-order shape function (a simplified model for translation) is used, as the method is inspired by particle image velocimetry; (ii) texture memory on GPU device is employed for interpolation of intensity at subpixel location.

Local DVC methods also get benefit from GPU parallel computing. The reported implementations are all developed with the path-independent strategy [27–29]. The integer-voxel displacement at each POI is estimated through CC-based algorithms and fed to various iterative subpixel DVC algorithms. Compared with the serial CPU implementation of same DVC method, the GPU implementation can gain an improvement in computation speed by up to dozens of times (https://en.wikipedia.org/wiki/Hyper-threading) [29].

Although GPU parallel computing has demonstrated a significant superiority in computation efficiency for iterative subpixel DIC/DVC over its multicore CPU counterpart [9,10,25,26,29], the heterogeneous parallel computing (HPC), which combines the strengths of GPUs and multicore CPUs, can be considered as a better solution due to the following reasons: (i) GPU cannot work independently in current computer architecture. Idling of multicore CPU when GPU is performing intensive computation is apparently a waste of resource; (ii) current high-performance CPU with up to 24 cores that support 48 parallel threads shows a computation efficiency comparable to middle-end GPU. A well designed HPC model including the two kinds of processing units provides the possibility to fully exploit the computer performance. Gates et al. [28] implemented a path-independent DVC method, which consists of FFT-CC algorithm for integer-voxel registration and Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for sub-voxel registration, on a cluster equipped with 120 nodes (each has two 6-core CPUs and three GPUs). In their implementation, the GPUs are used to estimate the normalized cross-correlation objective function and its gradient at POIs, and the multi-core CPUs handle the BFGS iteration as well as the integer-voxel registration. Their experiments show that the collaboration of 24 GPUs and 96 CPU cores can be 8 times faster than 96 CPU cores.

In this paper, the PiDIC method proposed in our previous study [15,25] is accelerated using HPC. The main differences between the study in ref. [28] and this paper can be summarized as following two points: (i) the implementation in ref. [28] is designed for cluster, whereas ours is oriented to a personal computing platform with single GPU and a multi-core CPU, which is more accessible to common DIC users; (ii) hybrid parallelism combining data parallelism and task parallelism is used in our HPC model, unlike the pure task parallelism employed in ref. [28], to achieve an optimized computation efficiency of GPU and CPU for DIC calculation. The construction of HPC model and its performance evaluation are explained in detail based on the experimental study.

2 Principle and implementation

2.1 Principle of PiDIC method

Figure 1 illustrates the principle of the PiDIC method. The integer-pixel displacement vector ($\boldsymbol{P}_0 = [u, 0, 0, v, 0, 0]^{\mathrm{T}}$, where u and v are the displacements along x-axis and y-axis, respectively) is estimated at each POI using the FFT-CC algorithm and then fed as the initial guess into the IC-GN algorithm to get subpixel deformation vector ($\boldsymbol{P} = [u, u_x, u_y, v, v_x, v_y]^{\mathrm{T}}$, where u_x, u_y, v_x, v_y represent the gradients of u-component and v-component, respectively). The iterative procedure of the IC-GN algorithm continues until any of the two convergence conditions is satisfied. In this study, the convergence conditions are set as (i) the incremental deformation vector ($\Delta\boldsymbol{P} = \left[\Delta u, \Delta u_x, \Delta u_y, \Delta v, \Delta v_x, \Delta v_y\right]^{\mathrm{T}}$) becomes less than 0.001 pixels; (ii) the maximum iteration number (20) is reached. Compared with path-dependent DIC method, which usually adopted an initial guess transfer scheme, the PiDIC method is completely immune to the issue of error spreading caused by the deformation discontinuity or invalid areas. Furthermore, the breakage of dependence between the POIs makes the PiDIC method very suitable for highly parallel processing. For more details about the principle, the readers can refer to ref. [15].

To avoid the redundant computation in the iterative procedure of the IC-GN algorithm, the gradient maps of the reference image (before deformation) are precomputed using the fourth-order central difference method, and a global look-up table of bicubic B-spline interpolation coefficients is built to speed up the construction of gray map in the warped target subset (after deformation). These operations are carried out at the beginning of the PiDIC method, hence called precomputation.

2.2 Programming models of GPU and CPU parallel computing

A brief introduction about the programming models of CPU parallel computing and GPU parallel computing as well as the mapping from the programming models to hardware are given in this section, which helps the readers understand the design and implementation of the heterogeneous parallel computing model.

In this study, CPU parallel computing is developed based on OpenMP, which is one of the SIMD (single instruction, multiple data) models. This model splits a problem into independent sub-problems that can be solved by threads in parallel. Each thread is mapped to a CPU core for execution. Benefiting from the Intel Hyper-Threading technology (HTT) [30], one physical CPU core can support two logical parallel threads. It means that on an HTT-available Intel CPU with n cores, up to $2n$ logical threads can be run simultaneously. Although HTT might not accelerate computational intensive applications or even cause performance degradation in case of cache access conflict (http://www.sem.org/dic-challenge/), the proposed PiDIC method, fortunately, is one of the applications that take substantial advantage of HTT. Our experiments show that the peak speed of PiDIC on CPU with HTT is approximately 30% higher than that without HTT.

GPU parallel computing is developed on CUDA. In this model, a problem is also split into independent sub-problems. These sub-problems are solved by blocks in parallel. Differing from CPU parallel computing, each sub-problem can be further split and solved by the synergic threads within a block. In the architecture of NVIDIA GPU, the blocks are mapped to the streaming multiprocessors (SMs) for execution, while each thread within these blocks is mapped to a streaming processor (SP) on the SMs. a SM can support multiple blocks simultaneously. The upper limit of block number on a SM depends on the number of registers and

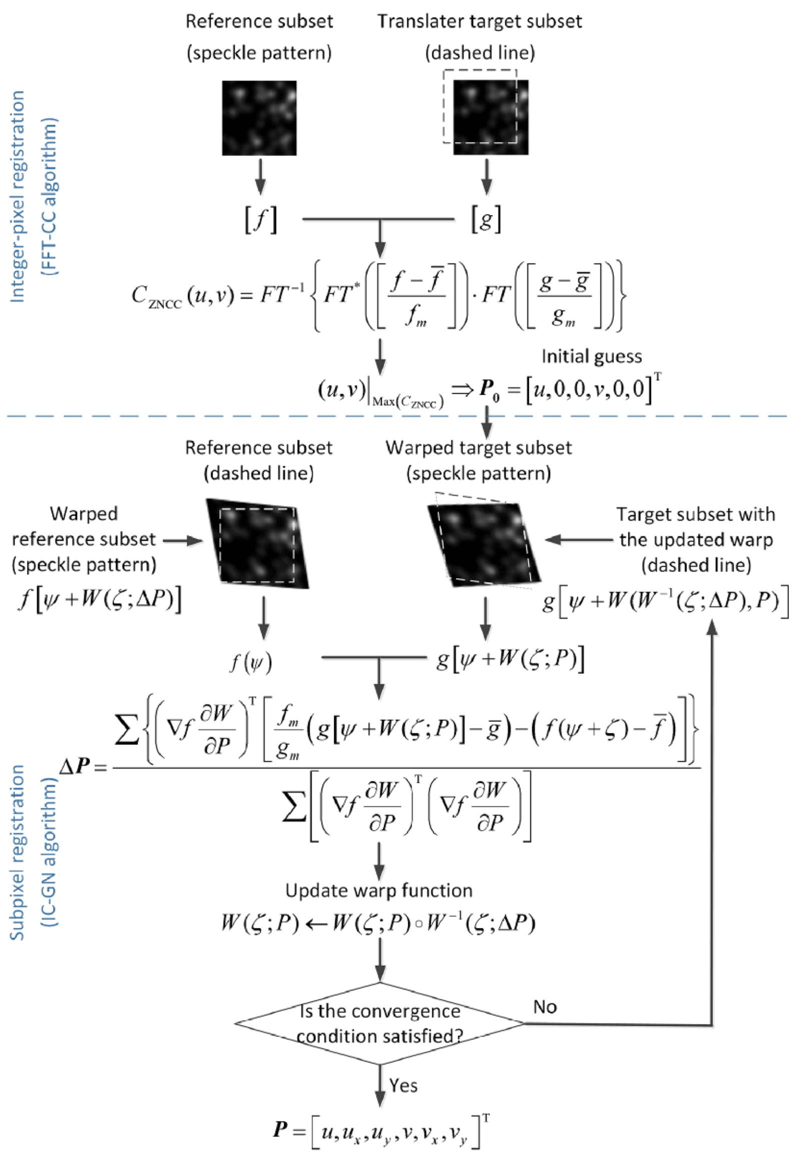
Integer-pixel registration (FFT-CC algorithm)
Reference subset (speckle pattern)
Translater target subset (dashed line)
$[f]$
$[g]$
$C_{\mathrm{ZNCC}}(u,v)=FT^{-1}\left\{FT^{*}\left(\left[\frac{f-\bar{f}}{f_m}\right]\right)\cdot FT\left(\left[\frac{g-\bar{g}}{g_m}\right]\right)\right\}$
Initial guess
$(u,v)|_{\mathrm{Max}(C_{\mathrm{ZNCC}})}\Rightarrow \boldsymbol{P}_0=[u,0,0,v,0,0]^{\mathrm{T}}$
Subpixel registration (IC-GN algorithm)
Reference subset (dashed line)
Warped target subset (speckle pattern)
Warped reference subset (speckle pattern)
$f[\psi+W(\zeta;\Delta P)]$
Target subset with the updated warp (dashed line)
$g[\psi+W(W^{-1}(\zeta;\Delta P),P)]$
$f(\psi)$
$g[\psi+W(\zeta;P)]$
$\Delta\boldsymbol{P}=\frac{\sum\left\{\left(\nabla f\frac{\partial W}{\partial P}\right)^{\mathrm{T}}\left[\frac{f_m}{g_m}(g[\psi+W(\zeta;P)]-\bar{g})-(f(\psi+\zeta)-\bar{f})\right]\right\}}{\sum\left[\left(\nabla f\frac{\partial W}{\partial P}\right)^{\mathrm{T}}\left(\nabla f\frac{\partial W}{\partial P}\right)\right]}$
Update warp function
$W(\zeta;P)\leftarrow W(\zeta;P)\circ W^{-1}(\zeta;\Delta P)$
Is the convergence condition satisfied?
No
Yes
$\boldsymbol{P}=[u,u_x,u_y,v,v_x,v_y]^{\mathrm{T}}$

Figure 1 (Color online) Principle of the PiDIC method.

shared memory available for allocation.

A GPU contains hundreds to thousands of SPs, whereas the number of cores on a CPU is usually less than thirty. However, the number of concurrent tasks on GPU is usually close to that on CPU, because a GPU block generally consists of tens or hundreds of threads, which limit the maximum number of active blocks on a SM. It should be mentioned that the performance gap between GPU and CPU for a randomly chosen algorithm, if its implementation is sufficiently optimized, may not be hundreds of times huge (as advertised in commercial propaganda) [31].

2.3 Heterogeneous parallel computing model for PiDIC

In a HPC model, GPU and CPU can be managed through task-parallelism or data-parallelism. In the former, the sub-tasks split from the major task are categorized according to their characteristics. GPU generally takes the computing-intensive sub-tasks, while CPU handles the ones containing complex logic instructions. In the latter, the data are split into two parts and assigned to GPU and CPU, which process the data with same instructions. As all the three stages in our PiDIC method (i.e. precomputation, integer-pixel registration and subpixel registration) are computing-intensive tasks, which can be processed several times faster on GPU than on CPU, the task-parallelism may not be a wise option. In fact, the HPC model for DVC in ref. [28] developed with a task-parallelism strategy on a super cluster with high performance computing cards (NVIDIA Tesla M2090) shows insignificant superiority in computation efficiency over a single GPU parallel computing model on a desktop computer equipped with a low-end graphics card (NVIDIA GTX 680) [29].

Figure 2 illustrates the HPC model proposed in this study, which uses a hybrid mode combing task-parallelism and data-parallelism. The precomputation stage is handled totally by GPU. This arrangement may avoid unnecessary memory access and communication between the processing units, since the precomputation can be finished very quickly by GPU, only occupies around 1% of the overall computation time, as discussion in Section 3.2. In the image registration stages, the POIs are split into two groups according to the computing performance of GPU and CPU. Then GPU tackles x% of POIs and CPU takes the rest.

Optimal proportion of POIs that assigned to the two kinds of processing units guarantees the high performance of the HPC model. This proportion depends on the computation efficiency of GPU and CPU. However, it is hard to construct a general model for the equivalent computation efficiency of GPU and CPU, which is influenced by various factors, including frequency, number of processing cores, instructions per cycle, bandwidth, and processor architecture etc. In the case of a specific computer, the optimal configuration of HPC is associated with the parameters of DIC method, which can be achieved through a series of trial tests.

Figure 3 shows the coarse-grained parallelism of the image registration on the two kinds of processing units. The number of active GPU blocks (or CPU threads) n, limited by the hardware, is far less than the total number of POIs m. In the implementation, each GPU block (or CPU thread) should process a queue of POIs in a serial fashion. If m is not an integer multiple of n, there will be k (k=m mod n) GPU blocks or CPU threads that need to process one more POI than the others.

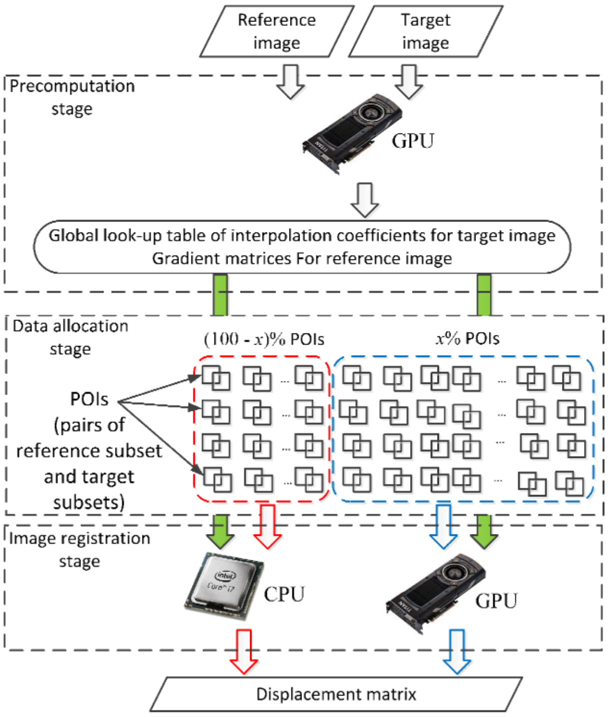
Reference image
Target image
Precomputation stage
GPU
Global look-up table of interpolation coefficients for target image
Gradient matrices For reference image
Data allocation stage
(100 - x)% POIs
x% POIs
POIs (pairs of reference subset and target subsets)
Image registration stage
CPU
GPU
Displacement matrix

Figure 2 (Color online) Illustration of heterogeneous parallel computing model with a hybrid mode of parallelism.

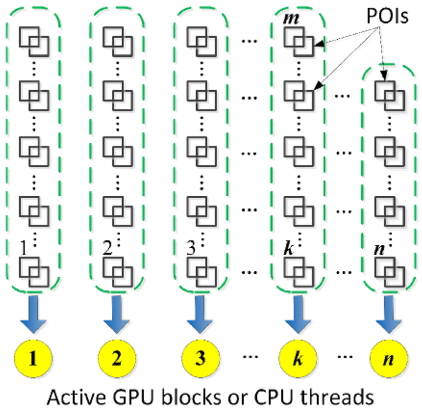

Figure 3 (Color online) Coarse-grained parallelism of image registration on GPU and CPU.

3 Experimental study

Figure 4 shows two grayscale speckle images used in the experiments. Figure 4(a) is a 1280×480-pixel speckle image of the epoxy plate surface (Example 1). It is translated in the Fourier domain according to the shift theorem [32] to generate twenty target images, with a pre-set subpixel displacement along the x-axis ranging from 0 to 1 pixel. The deformation between every two successive images is set to 0.05 pixels. Figure 4(b) shows the first speckle image of an image series appreciatively obtained from the website of the Society for Experimental Mechanics (Digital Image Correlation Challenge, Sample 12) (http://www.sem.org/dic-challenge/) (Example 2). The image series records the deformation of a plate specimen (with a round hole in the middle) subjected to unidirectional tension along x-axis. For the convenience of display in the paper, the original images with a size of 400×1040 pixels are rotated by 90° clockwise.

Figure 4(a) and (b) are set as the reference image for Example 1 and Example 2, respectively. The rectangular regions of interest are set as 960×320 pixels and 280×900 pixels, respectively.

The HPC model for the PiDIC method is programmed using C++ language. GPU parallel computing and CPU parallel computing are developed based on CUDA 6.5 and OpenMP 2.0, respectively. The number of threads in a GPU block is set as 128. A desktop computer equipped with an Intel i7-6950K CPU (3.0 GHz, 10 Cores and 20 Threads), 16 GB RAM and a NVIDIA K40c computing card (2880 CUDA cores, 876 MHz, 12 GB RAM, Bandwidth

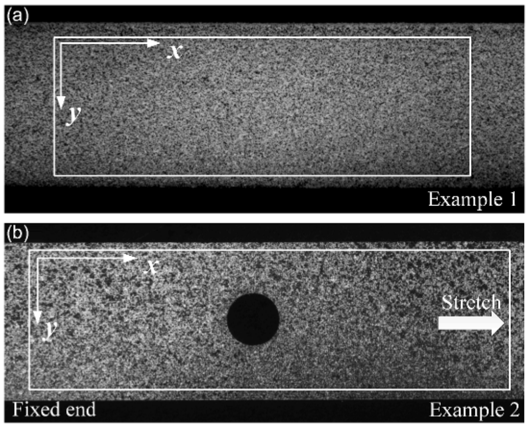

Figure 4 Two examples for experimental study: (a) a speckle image of epoxy plate, which is used as the reference image to generate a series images with translation along x-axis; (b) the first image of an image series recording the deformation of a CFRP plate with a round hole subjected to uniaxial tension along x-axis (obtained from the Society for Experimental Mechanics, http://www.sem.org/dic-challenge/). The solid-line rectangles outline the ROIs for DIC computation.

288 GB s^{-1}) is used as the computation platform for experimental study.

3.1 Single-precision floating-point versus double-precision floating-point

GPUs are designed to deliver optimal performance for processing of single-precision floating-point data. However, the high computation efficiency achieved with single-precision floating-point risks the accuracy of results due to the accumulation of round-off errors. Thus, a judicious choice of number format should be the first issue addressed in the development of GPU parallel computing. Unfortunately, there still lacks a systematic comparison heretofore between the two kinds of number formats in GPU-accelerated DIC.

Figure 5 compares the mean bias error (systematic error) and standard deviation error (random error) obtained on GPU and CPU for Example 1 (33×33-pixel subset, 15504 POIs), using single-precision floating-point format and double-precision floating-point format respectively. The errors obtained in the four conditions are nearly identical. The mean bias errors and standard deviation error are less than 0.0063 pixels and 0.0012 pixels, which fall in the acceptable range of experimental accuracy limits.

To compare further the results attained using the two kinds of number formats, two indicators (i.e., mean absolute deviation e_{A} and mean relative deviation e_{R}) are defined as

$$e_{\mathrm{A}} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\left(u_i^{\mathrm{s}} - u_i^{\mathrm{d}}\right)^2}, \qquad e_{\mathrm{R}} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\left(\frac{u_i^{\mathrm{s}} - u_i^{\mathrm{d}}}{u_i^{d}}\right)^2}, \tag{1}$$

where N is the POI number. u_i^{s} and u_i^{d} represent the u-component obtained at the ith POI using single-precision floating-point format and double-precision floating-point format, respectively.

Figure 6 shows the mean absolute deviation and mean relative deviation obtained on GPU and CPU. In Example 1, e_{A} stays at a stable level between 1.0×10^{-5} pixels and 1.2×10^{-5} pixels, which makes e_{R} decrease when the measured value of u-component becomes larger (Figure 6(a)). In most cases, e_{R} is below 0.01%. For u-component of 0.05 pixels, it is still less than 0.02%. In Example 2 (33×33-pixel subset, 12375 POIs), e_{A} obtained on GPU increases considerably from 1.1×10^{-5} pixels to 1.3×10^{-4}, whereas that on CPU is steadily below 1.0×10^{-4} pixels, as shown in Figure 6(b). The increase in e_{A} can be attributed to the larger deformation in Example 2, which is up to 12 pixels in the later stage. e_{R} attained for Example 2 is within 0.01% on GPU and CPU, even lower than that for Example 1.

It is noteworthy that GPU and CPU with double-precision floating-point format give identical results for the valid areas in both examples. Only at several POIs in Example 2, which fall in the middle hole (Figure 4(b)), GPU and CPU give quite different results because the IC-GN algorithm cannot converges to a specific value when dealing with those almost completely dark subsets. Thus, the data obtained at those POIs are excluded from the statistics. GPU with single-precision floating-point format seems to be more vulnerable to the round-off errors (Figure 6(b)). Nevertheless, a comparison between the results obtained on GPU and CPU with single-precision floating-point format for the two examples indicates that the relative difference could be negligible (less than 0.005%).

Figure 7 compares the computation time consumed for the two examples on GPU and CPU using the two kinds of number formats. In both examples, use of single-precision floating-point format leads to a significant increase in computation speed on GPU, compared with its double-precision counterpart. GPU with single-precision floating-point can be around 2.2 times faster than GPU with double-precision floating-point. However, single-precision floating-point does not speed up the calculation on CPU. It even slows down the program slightly (about 3%), since CPU treats the two kinds of number formats as double precision floating-point and the emulation of single precision floating-point costs extra time.

Based on the statistics mentioned above, it can be concluded that the DIC algorithm with single-precision floating-point achieves optimal computation efficiency on GPU at trivial cost of accuracy, while CPU with single-precision floating-point suffers slight loss of both efficiency and accuracy. To facilitate the programming, this number format is chosen in the HPC model for the operations of both GPU and CPU.

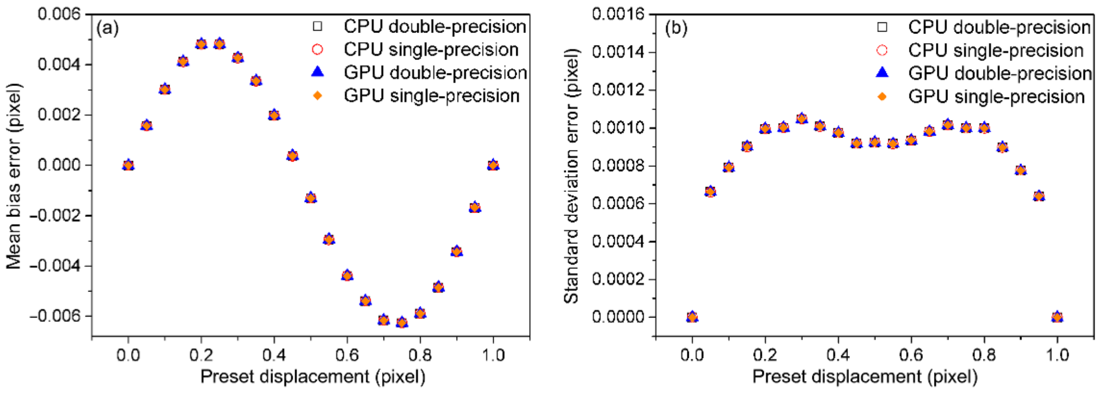

Figure 5 (Color online) (a) Mean bias error and (b) standard deviation error obtained on GPU and CPU with single-precision and double-precision floating-point formats for Example 1.

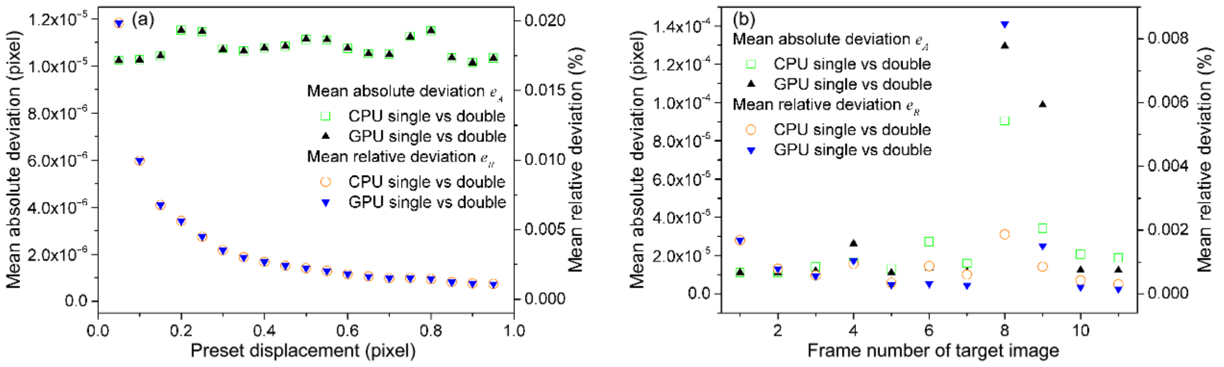

Figure 6 (Color online) Mean absolute deviation and mean relative deviation obtained on GPU and CPU with single-precision and double-precision floating-point formats for (a) Example 1 and (b) Example 2.

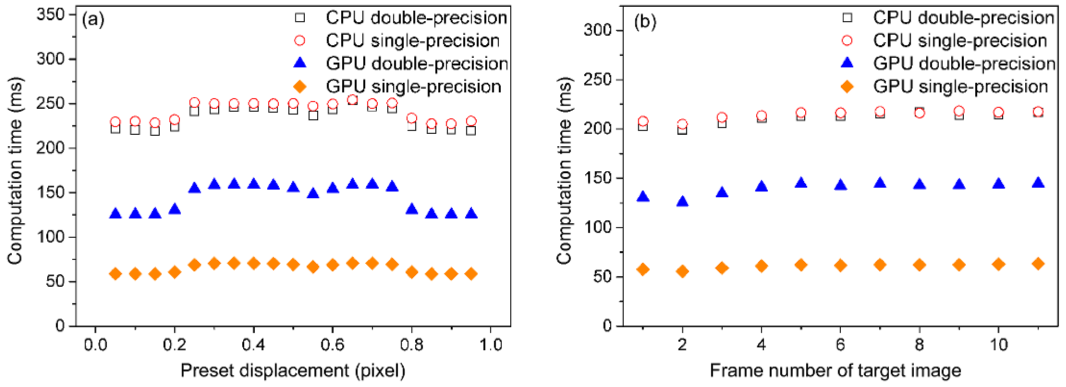

Figure 7 (Color online) Computation time consumed by GPU and CPU with single-precision and double-precision floating-point formats for (a) Example 1 and (b) Example 2.

3.2 Speed contest between GPU and CPU

Figure 8 shows the distribution of the computation time spent by GPU and CPU in the three stages (Example 2, 33×33-pixel subset, 20790 POIs). The overall computation time consumed by GPU is only 28% of that by CPU. In particular, the time share of the precomputation stage on GPU is much lower than that on GPU, which benefits from the fine-grained parallelism in GPU parallel computing. In this stage, the 128 threads in a GPU block are further divided into 8 groups (each contains 16 threads). Each group is employed to calculate the 16 interpolation coefficients for estimation of intensity at any subpixel location in a 2×2-pixel grid in the target image [13]. In contrast, in CPU parallel computing, the 16 interpolation coefficients for a 2×2-pixel grid are calculated by a thread in a serial fashion. The difference in parallelism makes GPU finish the precomputation stage in 1.38 ms, 15.7 times faster than CPU (21.68 ms). In the image

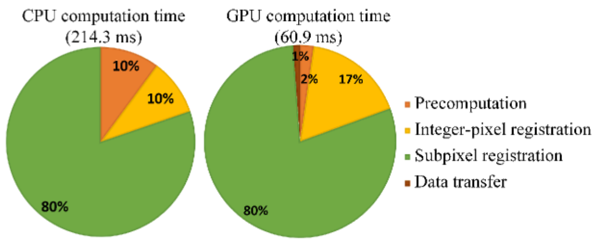

Figure 8 (Color online) Distribution of the computation time consumed by GPU and CPU in the three stages for Example 2.

registration stage, the fine-grained parallelism in GPU parallel computing also contributes its significant superiority over CPU parallel computing. Even the parallel degree of GPU and CPU are quite close at coarse-grained level, i.e. 22 on GPU and 20 on CPU respectively, GPU reaches a speed 3.3 times faster than CPU.

Figure 9 shows the flowcharts of the image registration stage on the two kinds of processing units. The integer-pixel and subpixel image registration at a POI is performed by one thread in CPU parallel computing, whereas in GPU parallel computing 128 threads in a block are involved in the same work. During the integer-pixel registration, cuFFT (a CUDA library for FFT) provides GPU the solution to calculate the cross-correlation function in fined-grained parallelism, and the position of maximum cross-correlation value can be located through parallel reduction method. In the most time-consuming subpixel registration, the fine-grained parallelism on GPU accelerates the construction of the warped target subset, which takes more than 70% of the computation time in this stage (Figure 10). Although there is only 6 of 128 threads that participate the rest part in the IC-GN algorithm and 122 threads are left idle, the waste of computation resource could be acceptably small. Arrangement of the construction of the warped target subset and the rest part with a task-parallelism strategy [28] may not leads to a better computation efficiency, since the saved time could be offset by the inter-block data exchange, as discussed in Section 2.3.

3.3 Parameter optimization of heterogeneous parallel model

Assignment of POIs to the two kinds of processing units plays a critical role in the HPC model to fully exploit the computing resource. An optimal proportion of POIs, i.e. GPU takes x% and CPU takes (100−x)%, should guarantee that GPU and CPU finish the processing task simultaneously. Thus, an optimal ratio can be defined as

$$\gamma=\frac{x}{100-x}=\frac{v_{\mathrm{GPU}}}{v_{\mathrm{CPU}}}, \qquad (2)$$

where v_{GPU} and v_{CPU} are computation speed of GPU and CPU in the stage of image registration, respectively. For specific hardware combination and DIC algorithm, the optimal ratio γ of HPC model is affected by the parameters of DIC calculation, i.e. subset size and POI number.

Figure 11 shows v_{GPU} and v_{CPU} achieved for Example 2, with

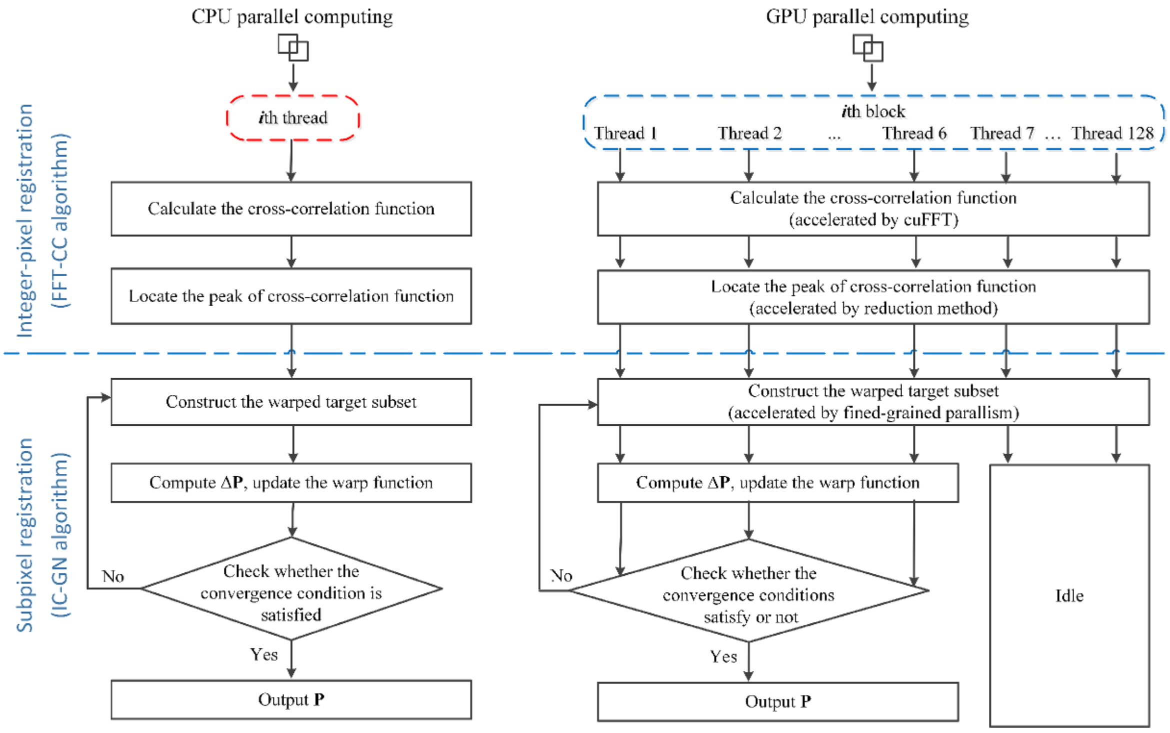
CPU parallel computing
GPU parallel computing
Integer-pixel registration
(FFT-CC algorithm)
Subpixel registration
(IC-GN algorithm)
ith thread
ith block
Thread 1
Thread 2
...
Thread 6
Thread 7
...
Thread 128
Calculate the cross-correlation function
Calculate the cross-correlation function (accelerated by cuFFT)
Locate the peak of cross-correlation function
Locate the peak of cross-correlation function (accelerated by reduction method)
Construct the warped target subset
Construct the warped target subset (accelerated by fined-grained parallism)
Compute ΔP, update the warp function
Check whether the convergence condition is satisfied
Check whether the convergence conditions satisfy or not
No
Yes
Output P
Idle

Figure 9 (Color online) Fine-grained parallelism in GPU parallel computing and CPU parallel computing in the image registration stage.

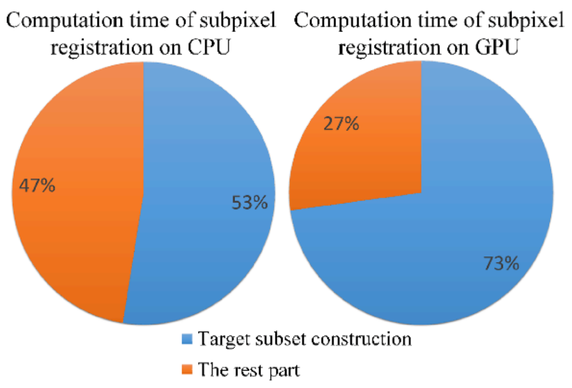

Figure 10 (Color online) Distribution of the computation time consumed by GPU and CPU in subpixel registration stage for Example 2.

POI number increasing from 120 to 20790 and commonly used subset sizes ranging from 17×17 pixels to 41×41 pixels. It is interesting that the computation speeds of both GPU and CPU are at a relatively low level when processing small numbers of POIs (e.g. 120 or 534). The computation speed of CPU boosts to a relatively high but stable level (relative fluctuation is less than 5%) when the POI number exceeds 1246, whereas GPU reaches its stable speed when the POI number exceeds 5340.

The observed sensitivity of computation speed to POI number can be attributed to the time consumed by the initialization procedure that is irrelevant to POI number. In CPU parallel computing, OpenMP needs to organize the computation resource and open the parallel region at the beginning of computation [34]. This part takes a specific amount of time, which occupies less and less share of the overall computation time with increasing POI number. When the time spent on the initialization procedure becomes negligible to the overall computation time, v_{CPU} reaches a stable level. In GPU parallel computing, the effect of the initialization prcedure (including organization of computation resource and transferring of data from host memory to shared memory) seems more marked because the computation is much faster than the memory access. Thus, GPU needs much more POIs to make v_{GPU} reach a stable level.

Based on this mechanism, the computation time t in the image registration stage can be divided into two parts: t_c irrelevant to POI number and $t_p(N)$ dependent on POI number N. Figure 12 gives the plots of t versus N on GPU and CPU. It can be observed that there is a quasi-linear relationship between t_p and N. Thus, a linear model of computation time can be proposed as

$$t = t_c + t_p(N) = t_c + N\Delta t, \tag{3}$$

where Δt is the time required to process a POI. The values of t_c and Δt can be estimated through linear fitting, as listed in Table 1. For both GPU and CPU, t_c increases considerably with increasing subset size in a non-linear manner, indicating a clear correlation of the initialization procedure with the subset size. According to eq. (3), the computation speed of GPU or CPU for any POI number can be estimated through a

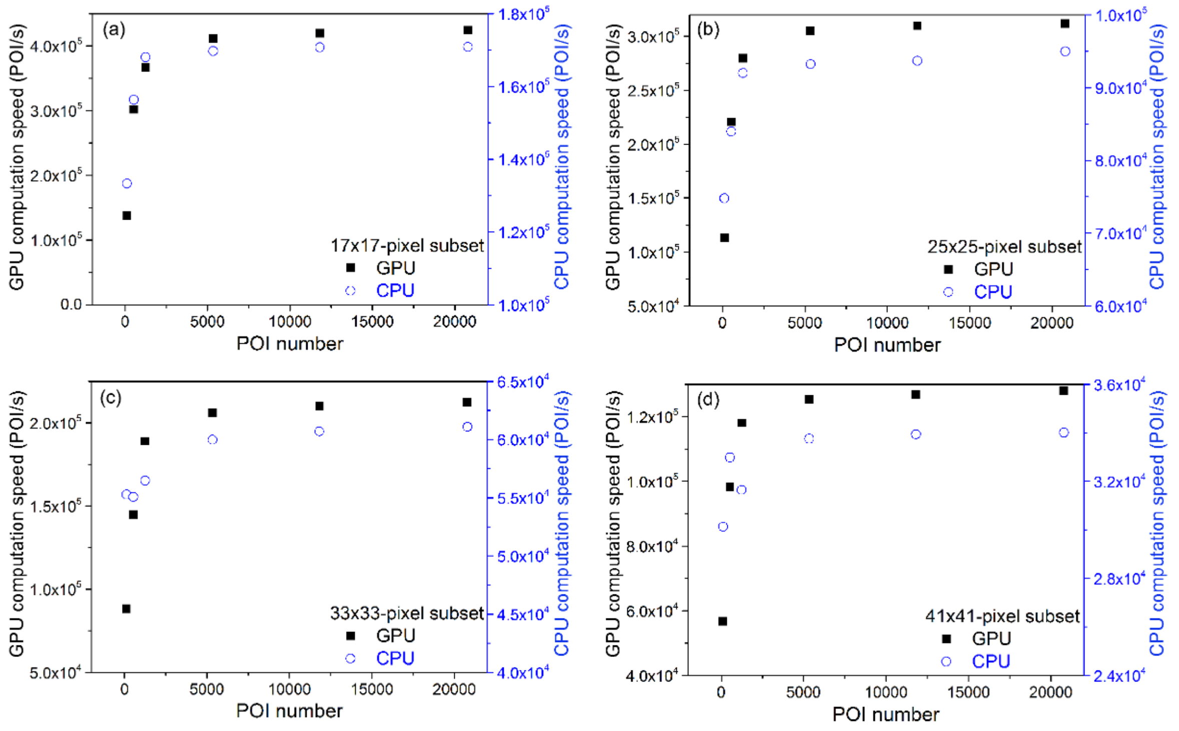

Figure 11 (Color online) Computation speed achieved by GPU and CPU for example 2 in processing various numbers of POIs with (a) 17×17-pixel subset, (b) 25×25-pixel subset, (c) 33×33-pixel subset and (d) 41×41-pixel subset.

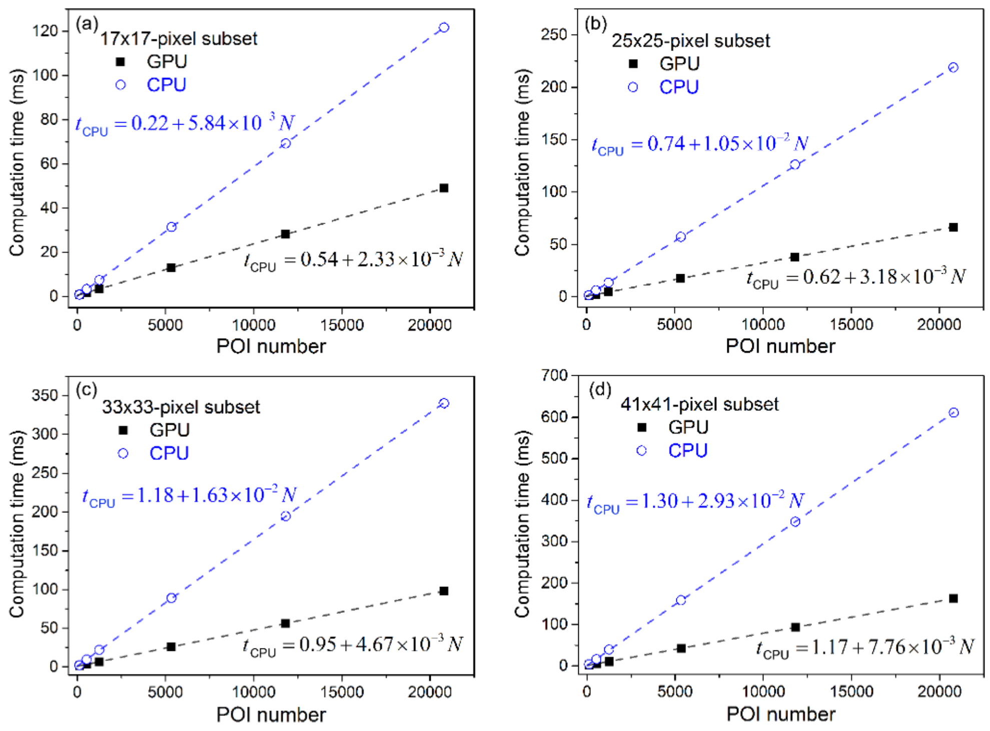

Figure 12 (Color online) Relation between POI number and the computation time achieved by GPU and CPU for Example 2 with (a) 17×17-pixel subset, (b) 25×25-pixel subset, (c) 33×33-pixel subset and (d) 41×41-pixel subset.

Table 1 Parameters (t_c and Δt) of computation time model for GPU and CPU

Subset size (pixel^2)	GPU		CPU	
	t_c (ms)	Δt (ms)	t_c (ms)	Δt (ms)
17×17	0.54	2.33×10^{-3}	0.22	5.84×10^{-3}
25×25	0.62	3.18×10^{-3}	0.74	1.05×10^{-3}
33×33	0.95	4.67×10^{-3}	1.18	1.63×10^{-3}
41×41	1.17	7.76×10^{-3}	1.30	2.93×10^{-3}

couple of trial calculations.

Based on the trial computation test, the profile of optimal ratio γ can be constructed as a function of subset size and computation time (Figure 13(a)), according to which the optimal computation speed attained by the HPC model can be estimated to make a performance profile (Figure 13(b)). In practical applications, these profiles for a specific computer can be obtained through a couple of trial computation tests using two of the acquired images, with several POI numbers and two or three subset sizes which may be used in the test.

3.4 Real-time DIC

In this section, a demonstration test that simulates the application of real-time DIC is carried out using example 2. If the images are recorded at a video rate (24 fps), the computation of each image pair in real-time DIC should be finished in 41.6 ms. According to the performance profile of the HPC model (Figure 13), the upper limit of POI number and optimal ratio γ can be estimated for the selected subset size. For instance, there can be up to 10353 POIs or 15542 POIs set in the image when using 33×33-pixel subset or 25×25-pixel subset, respectively. Optimal ratio γ can be set as 3.5 and 3.3 in the two cases. It is noteworthy that the upper limits of POI number are obtained according to the model constructed on the average computation time of the image series. In practical applications, the computation time consumed for every frame varies due to the changes in image quality or deformation feature. Thus, a conservative estimation, e.g. around 90% of the predicted maximum POI number, is chosen in the test, i.e. 9720 POIs and 14198 POIs, respectively.

Figure 14 compares the computation time spent on each image pair. The optimal ratio γ guarantees the time per frame satisfying the requirement of real-time DIC computation

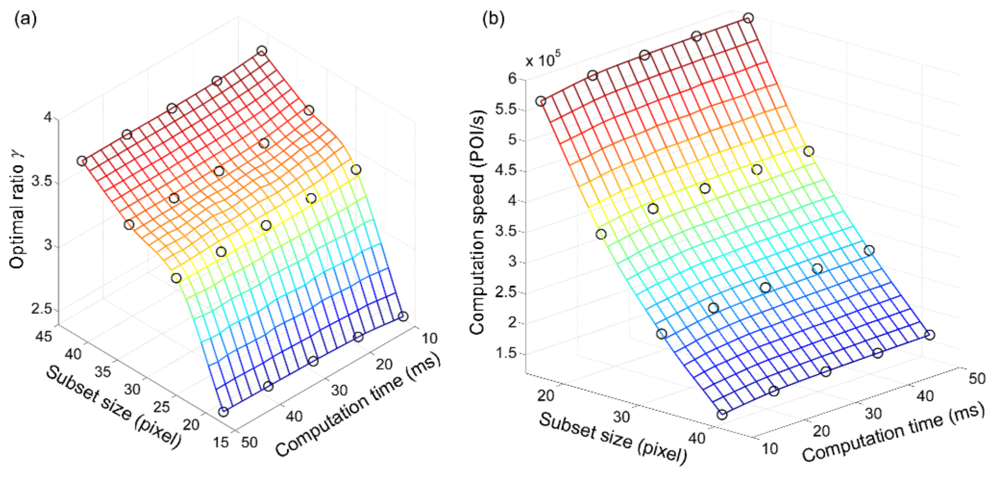

Figure 13 (Color online) Profiles of (a) optimal ratio and (b) predicted computation speed of the HPC model for Example 2.

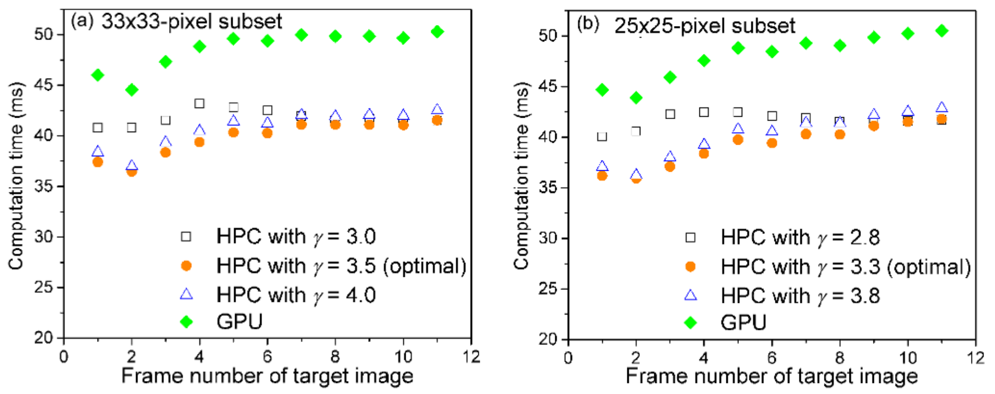

Figure 14 (Color online) Computation time per frame achieved by the HPC model using (a) 33×33-pixel subset and (b) 25×25-pixel subset.

(less than 41.6 ms), whereas a 15% variation of γ results in failure to achieve the real-time processing speed at some frames. It can also be found that HPC model is about 1.2 times faster than the mere GPU parallel computing, consistent with the performance evaluation of the two kinds of processing units.

4 Conclusions

A heterogeneous parallel computing model combing GPU and multicore CPU was developed for iterative subpixel DIC with a hybrid mode of task-parallelism and data-parallelism, based on a systematic comparison between GPU parallel computing and CPU parallel computing. The heterogeneous parallel computing demonstrates outstanding performance for the IC-GN algorithm-based subpixel DIC and makes the high accuracy real-time DIC feasible with high resolution. Some key conclusions can be summarized as follows:

(1) GPU parallel computing demonstrates much higher computation efficiency than CPU parallel computing for the iterative subpixel DIC of computing-intensive nature. This superiority is ascribed to the fine-grained parallelism in GPU parallel computing.

(2) A well designed HPC model can leads to further improvement of computation speed compared with GPU parallel computing. The optimization of the HPC model is highly case-specific. On a specific computer, the data allocation plays a critical role in fully exploiting the performance of HPC. The optimal ratio of the POIs assigned to GPU to those assigned to CPU can be estimated through a simple trial computation test.

(3) The HPC model shows lower computation efficiency when processing small numbers of POI due to the pre-computation and initialization of parallel computing, which consumes a certain amount of time. The share of the two parts in the overall computation time becomes smaller with increasing POI number, which raises the computation efficiency of the HPC model to a stable level. This factor should be considered in the application of HPC.

It is noteworthy that the GPU parallel computing technique is in an era of rapid development. Figure 15 shows the computation speed achieved on a newly released NVIDIA Titan Xp graphics card (3840 CUDA cores, 1582 MHz, 12 GB RAM, Bandwidth 548 GB s^{-1}) for Example 2. Although the Titan Xp card contains only 30% more CUDA

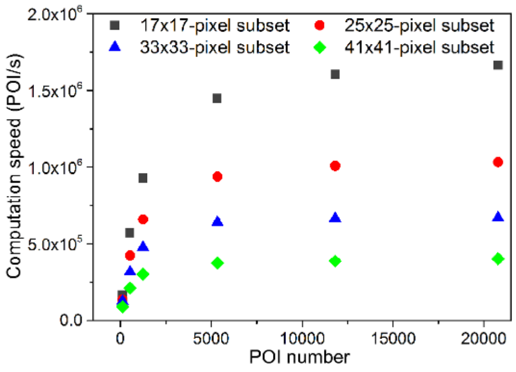

Figure 15 Computation time achieved by Titan Xp graphics card for Example 2.

cores than the Tesla K40c card, the state-of-the-art architecture makes the former 2.5 times faster than the latter. In this case, CPU parallel computing becomes an insignificant option in the HPC model. However, multicore CPUs still have the chance to make their exclusive contribution to the performance of the HPC model when some complicated processing strategies with substantial logic instructions are introduced to ameliorate the robustness of the DIC methods.

The work was supported by the National Natural Science Foundation of China (Grant Nos. 11772131, 11772132, 11772134 & 11472109), the Natural Science Foundation of Guangdong Province, China (Grant Nos. 2015A030308017, 2015A030311046 & 2015B010131009), the Opening fund of State Key Laboratory of Nonlinear Mechanics (LNM), CAS, and the State Key Lab of Subtropical Building Science, South China University of Technology (Grant Nos. 2014ZC17 & 2017ZD096).

1 Bing P, Xie H M, Xu B Q, et al. Performance of sub-pixel registration algorithms in digital image correlation. Meas Sci Technol, 2006, 17: 1615–1621
2 Tong W. Formulation of Lucas-Kanade digital image correlation algorithms for non-contact deformation measurements: A review. Strain, 2013, 49: 313–334
3 Tao G, Xia Z. A non-contact real-time strain measurement and control system for multiaxial cyclic/fatigue tests of polymer materials by digital image correlation method. Polym Test, 2005, 24: 844–855
4 Wu R, Kong C, Li K, et al. Real-time digital image correlation for dynamic strain measurement. Exp Mech, 2016, 56: 833–843
5 Sutton M A, Orteu J J, Schreier H. Image Correlation for Shape, Motion and Deformation Measurements: Basic Concepts, Theory and Applications. New York: Springer, 2009
6 Pan B, Li K, Tong W. Fast, robust and accurate digital image correlation calculation without redundant computations. Exp Mech, 2013, 53: 1277–1289
7 Baker S, Matthews I. Lucas-kanade 20 years on: A unifying framework. Int J Comp Vision, 2004, 56: 221–255
8 Baker S, Matthews I. Equivalence and efficiency of image alignment algorithms. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Kauai, 2001. 1090–1097
9 Shao X, Dai X, He X. Noise robustness and parallel computation of the inverse compositional Gauss-Newton algorithm in digital image correlation. Opt Laser Eng, 2015, 71: 9–19
10 Pan B, Tian L. Superfast robust digital image correlation analysis with parallel computing. Opt Eng, 2015, 54: 034106
11 Chen W, Jiang Z, Tang L, et al. Equal noise resistance of two mainstream iterative sub-pixel registration algorithms in digital image correlation. Exp Mech, 2017, 57: 979–996
12 Pan B, Li K. A fast digital image correlation method for deformation measurement. Opt Laser Eng, 2011, 49: 841–847
13 Pan Z, Chen W, Jiang Z, et al. Performance of global look-up table strategy in digital image correlation with cubic B-spline interpolation and bicubic interpolation. Theor Appl Mech Lett, 2016, 6: 126–130
14 Pan B. An evaluation of convergence criteria for digital image correlation using inverse compositional Gauss-Newton algorithm. Strain, 2014, 50: 48–56
15 Jiang Z, Kemao Q, Miao H, et al. Path-independent digital image correlation with high accuracy, speed and robustness. Opt Laser Eng, 2015, 65: 93–102
16 Gao W, Kemao Q. Parallel computing in experimental mechanics and optical measurement: A review. Opt Laser Eng, 2012, 50: 608–617
17 Wang T, Qian K. Parallel computing in experimental mechanics and optical measurement: A review (II). Opt Laser Eng, 2017, 50: 608–617
18 Pratx G, Xing L. GPU computing in medical physics: A review. Med Phys, 2011, 38: 2685–2697
19 Navarro C A, Hitschfeld-Kahler N, Mateu L. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. Commun Commut Phys, 2014, 15: 285–329
20 Leclerc H, Périé J N, Roux S, et al. Integrated digital image correlation for the identification of mechanical properties. In: Gagalowicz A, Philips W, eds. Computer Vision/Computer Graphics Collaboration Techniques. Berlin: Springer, 2009. 161–171
21 Leclerc H, Périé J N, Hild F, et al. Digital volume correlation: What are the limits to the spatial resolution? Mech Indust, 2012, 13: 361–371
22 Gembris D, Neeb M, Gipp M, et al. Correlation analysis on GPU systems using NVIDIA's CUDA. J Real-Time Image Proc, 2011, 6: 275–280
23 Marciniak B, Marciniak T, Lutowski Z, et al. Usage of digital image correlation in analysis of cracking processes. Image Proc Commun, 2013, 17: 21–28
24 Singh A, Omkar S N. Digital image correlation using GPU computing applied to biomechanics. Biomed Sci Eng, 2013, 1: 1–10
25 Zhang L, Wang T, Jiang Z, et al. High accuracy digital image correlation powered by GPU-based parallel computing. Opt Laser Eng, 2015, 69: 7–12
26 Le Besnerais G, Le Sant Y, Lévêque D. Fast and dense 2D and 3D displacement field estimation by a highly parallel image correlation algorithm. Strain, 2016, 52: 286–306
27 Bar-Kochba E, Toyjanova J, Andrews E, et al. A fast iterative digital volume correlation algorithm for large deformations. Exp Mech, 2015, 55: 261–274
28 Gates M, Heath M T, Lambros J. High-performance hybrid CPU and GPU parallel algorithm for digital volume correlation. Int J High Perform Comput Appl, 2015, 29: 92–106
29 Valle V, Hedan S, Cosenza P, et al. Digital image correlation development for the study of materials including multiple crossing cracks. Exp Mech, 2015, 55: 379–391
30 Wang T, Jiang Z, Kemao Q, et al. GPU accelerated digital volume correlation. Exp Mech, 2016, 56: 297–309
31 Leng T, Ali R, Hsieh J, et al. An empirical study of hyper-threading in high performance computing clusters. Linux HPC Revolution, 2002
32 Lee V W, Hammarlund P, Singhal R, et al. Debunking the 100X GPU vs. CPU myth. SIGARCH Comput Archit News, 2010, 38: 451–460
33 Schreier H W, Braasch J R, Sutton M A. Systematic errors in digital image correlation caused by intensity interpolation. Opt Eng, 2000, 39: 2915–2921
34 Lustig D, Martonosi M. Reducing GPU offload latency via fine-grained CPU-GPU synchronization. In: Procedding of IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). Washington: IEEE Computer Society, 2013. 354–365