Towards high-speed optical measurement of shape and deformation

Wang, Tianyi

2018

Wang, T. (2018). Towards high-speed optical measurement of shape and deformation.
Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73234>

<https://doi.org/10.32657/10356/73234>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**TOWARDS HIGH-SPEED OPTICAL
MEASUREMENT OF SHAPE AND
DEFORMATION**

WANG TIANYI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2018

TOWARDS HIGH-SPEED OPTICAL
MEASUREMENT OF SHAPE AND DEFORMATION

WANG TIANYI

2018

WANG TIANYI

TOWARDS HIGH-SPEED OPTICAL MEASUREMENT OF SHAPE AND DEFORMATION

WANG TIANYI

School of Computer Science and Engineering

A thesis submitted to Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2018

Acknowledgment

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Qian Kemao for the continuous support of my PhD study and research, for his patience, motivation, and enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I am indebted to my co-supervisor Prof. Seah Hock Soon, for his kind advice, encouragement, and helpful criticism in my write-up. His constructive comments on my work lit up the way of my research and helped me successfully complete this thesis. Special gratitude goes out to Prof. Lin Feng, who enlightened me on doing research and provided me the valuable opportunity to pursue my PhD in this world-renowned university. Thanks also go to my thesis advisory committee member Prof. Lee Yong Tsui, for his inspiration, insightful comments, and hard questions.

I am grateful to my my parents, who have provided me through moral and emotional support in my life. Special thanks go to my girl friend, Fu Yue, a beautiful and optimistic lady, who has accompanied me in Singapore along the way.

And finally, last but by no means least, also to the beautiful music provided by the Study Music Project, which helped me keep calm and retain a positive attitude.

Contents

Summary	i
List of Figures	iv
List of Tables	vii
List of Algorithms	ix
1 Introduction	1
1.1 Shape and deformation	1
1.1.1 Computer simulation methods	1
1.1.2 Computer-vision based methods	3
1.2 Optical measurement techniques	4
1.3 Digital image correlation	6
1.3.1 Integer-pixel estimation	8
1.3.2 Sub-pixel registration	9
1.3.2.1 Forward additive Newton Raphson algorithm	9
1.3.2.2 Inverse compositional Gauss-Newton algorithm	10
1.3.2.3 Initial Gauss Transferring Schemes	12
1.3.2.4 Interpolation	13
1.3.3 Finite element-based global digital image correlation	15
1.4 Fringe pattern analysis and dynamic phase retrieval	16
1.4.1 Fringe pattern sequence model	16
1.4.2 Phase-shifting methods	17
1.4.3 Transform-based methods	18
1.4.4 Reference-based methods	19

1.5	Contributions and organization	20
1.5.1	Objective	20
1.5.2	Contributions	21
1.5.3	Organization	22
2	Literature review: CPU and GPU parallel computing in experi-	
	mental mechanics and optical measurement	24
2.1	Parallel computing on multi-core CPUs and GPUs	26
2.1.1	CPU and GPU architectures	26
2.1.2	Compute unified development architecture (CUDA)	28
2.2	Common parallel patterns	31
2.2.1	Pixelwise pattern	31
2.2.2	Tiling pattern	32
2.2.3	Divide-and-conquer pattern	32
2.2.4	Rendering and interpolation pattern	33
2.3	Existing parallel computing libraries	33
2.4	Applications in experimental mechanics and optical measurement .	34
2.4.1	Digital image correlation	34
2.4.1.1	Subset-based local DIC&DVC algorithms	34
2.4.1.2	FEM-based global DIC&DVC algorithms	37
2.4.2	Fringe pattern analysis	37
2.4.2.1	Fringe projection profilometry	37
2.4.2.2	Other fringe pattern analysis techniques	39
2.4.3	Tomography	40
2.4.3.1	Optical coherence tomography	41
2.4.3.2	Other tomography techniques	43
2.4.4	Hyperspectral imaging	44
2.4.4.1	Compression	44
2.4.4.2	Unmixing	45
2.4.4.3	Anomaly detection	46

2.4.4.4	Image classification	46
2.4.4.5	On-board unmixing	46
2.4.5	Computer generated holograms	46
2.4.6	Integral imaging	47
2.5	Discussion	47
3	Parallel digital image correlation	50
3.1	Introduction	50
3.2	Principle of the paDIC algorithm	51
3.3	Implementation of the paDIC algorithm	53
3.4	Experimental study	53
3.4.1	Verification of measurement accuracy	53
3.4.2	Evaluation of computation efficiency	55
3.5	Conclusion	58
4	Parallel digital volume correlation	60
4.1	Introduction	61
4.2	Principle of the paDVC algorithm	63
4.2.1	Integer-voxel estimation using 3D FFT-CC	63
4.2.2	Sub-voxel registration using 3D IC-GN	65
4.3	Implementation of the paDVC algorithm	67
4.3.1	Precomputation	67
4.3.2	Parallel 3D FFT-CC	70
4.3.3	Parallel 3D IC-GN	72
4.3.4	Batch processing scheme	72
4.4	Experimental study	75
4.4.1	Experimental specification	75
4.4.2	Verification on accuracy and precision	75
4.4.3	Verification on computation efficiency	76
4.5	Conclusion	81

5	Development of a hybrid CPU-GPU real-time digital image correlation system	82
5.1	Introduction	82
5.2	Real-time DIC system framework	85
5.2.1	Analysis of computational performance of FFT-CC and IC-GN on the CPU and the GPU	85
5.2.2	Three variations of the real-time DIC system framework . .	87
5.2.2.1	Variation 1: small-scale per-frame DIC system . . .	88
5.2.2.2	Variation 2: large-scale per-frame DIC system . . .	89
5.2.2.3	Variation 3: DIC system for ultra-fast deformation monitoring with sub-pixel compensation	90
5.2.3	Reference updating scheme	91
5.3	Example applications	92
5.4	Conclusion	95
6	Parallel reference-based dynamic phase retrieval	96
6.1	Introduction	96
6.2	Principle of LS3U	98
6.2.1	Least-squares fitting for phase change estimation	98
6.2.2	WFF for noise elimination	100
6.2.3	LS3U algorithm	100
6.3	GPU-powered real-time LS3U	101
6.3.1	Parallel computing strategies applied to the least-squares fitting algorithm	104
6.3.1.1	Data transfer	104
6.3.1.2	Parallelization of the pointwise pattern	104
6.3.1.3	Parallelization of the tiling pattern	104
6.3.1.4	Reduce redundant computations	105
6.3.1.5	Equation solving	106
6.3.2	Parallel computing strategies applied to the WFF algorithm	106

6.3.2.1	Precomputation of FFT of the windowed Fourier basis function	106
6.3.2.2	Precomputation of the Look-up table for FFT pre- ferred sizes	107
6.4	Experimental verification	108
6.4.1	Dynamic fringe projection	108
6.4.2	Dynamic speckle shearographic interferometry	110
6.4.3	Discussion	112
6.5	Conclusion	112
7	Conclusion and future work	114
7.1	Summary	114
7.2	Future work	116
7.2.1	Further improve the computation efficiency of DIC and DVC	116
7.2.2	Resolve deformation at boundaries and edges	117
7.2.3	Accurate and robust 3D real-time DIC system	120
7.2.4	Hybrid CPU and GPU applied to fringe pattern analysis . .	121
	References	122
	Publication	157

Summary

Optical measurement techniques, with the advantages of non-contact, non-destructiveness, and high accuracy and sensitivity, have been successfully developed and applied to study shapes and deformations of objects. Digital image correlation (DIC) and fringe pattern analysis (FPA) are two important categories of the optical measurement methods. DIC experiments can be performed in almost any imaging technology, while white-light optics are predominantly used. The displacement or strain field is measured by correlation analysis on a random speckle pattern on the surface of an object. On the contrary, FPA is based on regular fringe patterns that are generated by the well-known interference phenomenon. The shape of an object is obtained by phase extraction and phase unwrapping processes.

In the past decade, both DIC and FPA had made great progress. However, two key issues still need further improvement. First, in pursuit of higher resolutions for higher measurement accuracy, their computation burden has dramatically increased, making them hardly applied to study dynamic phenomena or integrated into real-time systems. Second, decorrelation is a serious issue when the deformation between the initial and the deformed configurations of an object is large. In fact, these two issues can be remedied by one single solution: increasing the computation efficiency, so that the speed requirement in dynamic experiments can be fulfilled. Large deformation can also be resolved by a high processing speed such that it is possible to insert multiple intermediate frames to make the inter-frame deformation small. In recent years, heterogeneous parallel computing platforms composing of CPUs and Graphics Processing Units (GPUs) have been widely used

to increase the computation efficiency of optical measurement techniques due to their cost-effectiveness, easy programming interface, and great portability and scalability, yet details of the employed parallel computing strategies have not been well explained. Therefore, in this thesis, a systematic study of powering optical measurement techniques, especially the DIC and FPA techniques, using CPU and GPU parallel computing are proposed.

First, the CPU and GPU parallel computing strategies applied to optical measurement methods, including digital image/volume correlation, fringe pattern analysis, tomography, hyperspectral imaging, computer-generated holograms (CGH), and integral imaging in the past five years are reviewed. It is found that high parallelism with the common four parallel patterns can be always exploited from these methods.

Second, a GPU powered high-accuracy parallel DIC (paDIC) algorithm is then proposed. It utilizes a path-independent initial guess transferring scheme so that the calculation at one point of interest (POI) is independent from the others. A 57.5+ times speedup compared with the sequential implementation of the same algorithm has been achieved. It is the fastest DIC algorithm using the 1st order (more accurate than the 0th order) shape function reported in the literature.

Third, the paDIC is extended to 3D digital volume correlation (DVC) to study the internal deformation of an object, and a parallel DVC (paDVC) is proposed. Due to the introduction of an extra dimension, paDVC is more complex than paDIC. Also, to efficiently process a large amount of 3D data volumes, a batch processing scheme is proposed and employed. The proposed paDVC algorithm is 23.3 times and 3.7 times faster than its sequential and CPU multi-core implementations, respectively. To our best knowledge, this is the fastest DVC algorithm reported so far yet without losing accuracy.

Fourth, based on the above two works, a real-time DIC system framework, which combines the strength of both the CPU and the GPU is proposed. While graphical user interface (GUI), data acquisition, processing, and display are pipelined on the CPU, the computationally intensive tasks are off-loaded to the

GPU. With the flexibly designed three variations of the framework, a real-time frame rate ranging from 30fps (frames per second) to 130fps has been achieved. Also, a reference updating scheme is employed to compensate the decorrelation when the deformation becomes large.

Finally, a real-time reference-based dynamic phase retrieval algorithm called G-LS3U is proposed to extract phase distributions from fringe or speckle patterns. Different parallel computing strategies are developed and applied to both the least-squares fitting and the windowed Fourier filtering (WFF) processes. G-LS3U achieved a remarkably high processing rate at 131+ fps, making G-LS3U the fastest reference-based dynamic phase retrieval algorithm reported heretofore.

Key Words: Optical measurement, multi-core, GPU, CUDA, parallel computing, digital image correlation, digital volume correlation, fringe pattern analysis, phase retrieval

List of Figures

1.1	Simulation of a beam shows inflated artifacts in (a) of the linear FEM under large rotation; the correct deformation is shown in (b) .	2
1.2	From left to right: large-scale deformation example interpolating mocap markers, full result after example-based fine-scale correction, the same result with full shading, and comparison to the real actors face.	4
1.3	Patterns used to carry the information of deformation: (a) DIC uses random speckle patterns; (b) FPA uses regular fringe patterns. . . .	6
1.4	Terms used in DIC. ROI: region of interest; POI: point of interest. .	7
1.5	Schematic of the calculation of bicubic interpolation coefficients for a sub-pixel position $C(x, y)$	14
1.6	Global DIC using eight-noded (Q8) elements to discretize the boundary positions.	15
1.7	An example of dynamic phase retrieval for speckle sherography: (a)-(c) speckle correlation fringe patterns at three representative frames; (d)-(f) the corresponding retrieved wrapped phase maps.	17
2.1	Schematically illustration of the architecture of the CPU.	27
2.2	Schematically illustration of the architecture of the GPU.	28
2.3	Overview of the CUDA programming model and its mapping to the NVIDIA GPU hardware. SM: Streaming Multiprocessors.	29
3.1	Schematically illustration of the principle of the paDIC algorithm combining the FFT-CC algorithm and the IC-GN algorithm.	51

3.2	Schematically illustration of the implementation of the paDIC. . . .	52
3.3	A simulated speckle pattern used as the reference image in numerical experiments.	54
3.4	Mean absolute errors (a) and standard deviations (b) of the calculated u -components.	55
3.5	A series of real images recording a plate specimen with a hole subjected to unidirectional tension along x-axis.	56
3.6	Displacement field along x-axis calculated using paDIC on the real images.	57
3.7	Comparison of computation speed between paDIC and seDIC. . . .	57
4.1	Schematic of the principle of DVC.	62
4.2	Flow chart of the proposed paDVC, where R and T refer to the reference sub-volume and target sub-volume, respectively, ξ is the local coordinates of points within the sub-volume.	64
4.3	Diagram of the tri-cubic interpolation at a sub-voxel position surrounded by 8 integer voxels.	69
4.4	Schematically illustration of the tiled computation for volume image gradients.	69
4.5	Flow chart of the parallel reduction algorithm within an individual block.	71
4.6	Diagram of the parallel 3D IC-GN algorithm.	73
4.7	Schematic of the batch processing scheme used in paDVC.	74
4.8	Computer simulated reference volume image $R(x, y, z)$ and the 10 translated target volume images from $T_1(x, y, z)$ to $T_{10}(x, y, z)$	76
4.9	Mean-bias errors (a) and standard deviation (b) of the measured w -components on noise-contaminated volumes calculated by paDVC using different subvolume sizes.	77
4.10	Mean-bias errors (a) and standard deviation (b) of the measured w -components on noise-free volumes calculated using different sub-volume sizes.	78

5.1	Comparison of the frame rate per second (fps) of (a) the FFT-CC and (b) the IC-GN algorithms implemented by the CPU using 2 cores and 6 cores, and the GPU on different number of POIs.	86
5.2	Schematic of a general pipelined DIC system framework.	88
5.3	Schematic of the small-scale per-frame DIC system framework. . . .	89
5.4	Schematic of the large-scale per-frame DIC system framework. . . .	89
5.5	Schematic of the DIC system framework for ultra-fast deformation monitoring.	90
5.6	Schematic of the reference updating scheme where the reference image is updated every m frames.	91
5.7	Parameters setting dialog of the proposed real-time DIC system, where the three variations of the framework can be selected in the right panel.	93
5.8	Visualization of the horizontal and vertical displacement fields calculated using the proposed real-time DIC system: (a-d) represent the horizontal and vertical displacement fields of POIs at frames 0, 24, 40, and 101, respectively	94
5.9	The screen-shots of the iDIC application operated on an iPhone 5s. . . .	94
6.1	Flowchart of the G-LS3U algorithm.	102
6.2	Schematic of the shifting process.	107
6.3	LS3U for dynamic fringe projection profilometry.	109
6.4	LS3U for dynamic speckle shearographic profilometry.	112
7.1	Schematic illustration of the non-central DIC method in dealing with the boundary deformation. If the original POI (the red point) falls outside the ROI, it will be moved to a new point (the blue point) on the intersection edge of the ROI.	118
7.2	Schematic illustration of the hybrid CPU and GPU system framework for fringe pattern analysis.	120

List of Tables

1.1	Statistics of computation time in different stages for the seDIC and the paDIC.	5
2.1	Summerization of the reviewed research works in DIC&DVC, FPA, tomography, HSI, CGH, and II.	49
3.1	Statistics of computation time in different stages for the seDIC and the paDIC.	58
4.1	Comparison of the average consumed time and computation speed among paDVC, muDVC, and seDVC.	79
4.2	Comparison of the average computation time consumed per batch in the precomputation step among the paDVC, the muDVC and the seDVC.	80
4.3	Comparison of computation efficiency for 3D IC-GN algorithm between paDVC, muDVC, and seDVC.	80
5.1	Comparison of the running time of the GPU-accelerated and the CPU sequential implementations of paDIC performed on different numbers of POIs on an iPhone 5s.	95
6.1	Comparison of the average running time and frame rates among the MATLAB, the CPU Multi-Core, and the proposed GPU-Based implementations of LS3U of the first example.	110
6.2	Average running time of each step of G-LS3U on one image frame of the first example.	111

6.3	Comparison of the average computation time of solving the linear system (Eq. 6.4) and WFF among the MATLAB, the CPU Multi-Core, and the proposed GPU-Based implementations on the first example.	111
-----	--	-----

List of Algorithms

6.1	Sequential LS3U algorithm.	101
6.2	Sequential LS3U_per_frame algorithm.	101
6.3	Parallelized LS3U_per_frame algorithm.	103

Chapter 1

Introduction

1.1 Shape and deformation

Shape and deformation of a solid are two important physical quantities that are commonly studied in the areas of experimental mechanics [1], material science [2], bio-mechanics [3], and computer simulation and animation [4]. Basically, there are two categories of methods of studying the shape and deformation of a solid: (i) via computer simulation based on continuum mechanics [5]; (ii) via measurement from real-world objects [6–8].

1.1.1 Computer simulation methods

In computer simulation methods, a constitutive model [9] is firstly required to define the relationship between the two physical quantities, i.e. strain and stress for a specific material, and approximate the response of the material to the exerted forces or stimuli. As an example, the 3D linear elasticity model is described as

$$\boldsymbol{\sigma} = \mathbf{E} : \boldsymbol{\epsilon} \quad \text{or} \quad \sigma_{ij} = E_{ijkl}\epsilon_{kl}, \quad (1.1)$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are the stress and the strain, respectively; $\mathbf{E}$ is a tensor composed of Young's modulus of elasticity and Poisson's ratio. Afterwards, the dynamic or the deformation of the object is governed by Newton's 2nd law as the following ordinary differential equation (ODE)

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{F}(t), \quad (1.2)$$

where $\mathbf{q}$ is the position of all the material points of the object; $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are their corresponding velocity and acceleration, respectively; $\mathbf{M}$ denotes their masses; $\mathbf{C}$ is the damping matrix; and $\mathbf{K}$ is the stiffness matrix.

Figure 1.1: Simulation of a beam shows inflated artifacts in (a) of the linear FEM under large rotation; the correct deformation is shown in (b)

The unknown in Eq. 1.2 is $\mathbf{q}$, which is a continuous vector field. The solution of $\mathbf{q}$ thus cannot be written in a compact form since it contains an uncountable set of vectors. To solve Eq. 1.2, finite element method (FEM) is a commonly applied discretization scheme where the continuous vector field can be described by a finite number of values. A numerical solver, for example, the gradient descent, the conjugate gradient, and the Newton Raphson, etc., is applied to perform the integration. It is worth noting that, however, three disadvantages restrict a much wider application of the computer simulation methods.

1. Passive control. The shape and deformation of a solid can only be passively obtained from the solutions of Eq. 1.2. Every time when a new shape or deformation is required, a new set of parameters, i.e. $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ have to be carefully designed and tuned.

2. Material properties are difficult to tune. Every material has its own properties, like Young's modulus, Poisson's ratio, damping factor, elasticity, plasticity,

etc. Also, these properties are highly influenced by the external environment. Thus, it is always complicated to tune them as well as choose an appropriate constitutive model for a material under a certain simulation context to fit them. The difficulty is even more serious for composite materials.

3. High computation complexity. To make the simulation be asymptotic to a real object, a large number of finite elements are required for a well discretization. Furthermore, to simulate complex deformation (i.e. large rotation and significant tension or compression), non-linear models should be used instead of linear ones. Otherwise, the incorrect deformation as shown in Fig. 1.1 (a) may occur while the correct one is shown in Fig. 1.1 (b). To simulate large elastic deformation, for instance, the non-linear St. Venant Kirchhoff model as well as the second order hexahedra are usually used as the constitutive model and the finite elements, respectively. In such case, a non-linear solver, e.g. the Newton Raphson should also be employed. Even worse, these non-linear processes can hardly be accelerated by parallel computing due to insufficient parallelism in the algorithms.

In computer graphics, the complexities mentioned above have been compensated by either using a co-rotated linear FEM method [10, 11] or reducing the dimension of non-linear FEM models by modal analysis [12–14]. Better deformation design [15] and editing [16] techniques have also been proposed. However, these improvements compromise with a loss of accuracy, leading them to be only applicable to computer animation and entertainment applications.

1.1.2 Computer-vision based methods

To alleviate the difficulties in parameter tuning, computer-vision based methods, which directly measure and reconstruct the shape and deformation from real-world objects using computer-vision techniques, have emerged [8]. Figure 1.2 shows a typical application for obtaining facial expression, in which some markers are put on the face of a man. The deformation of the markers is then captured by a motion-capture (mocap) technique. Since the deformation is obtained from real objects, it can be utilized as an output of Eq. 1.2 based on a certain constitutive model.

Therefore, the correct parameters can always be calculated by a fitting process [8] using multiple measured deformation examples.

Figure 1.2: From left to right: large-scale deformation example interpolating mocap markers, full result after example-based fine-scale correction, the same result with full shading, and comparison to the real actors face. Image courtesy of [17]

Although these computer-vision based techniques eliminate the difficulties in parameter tuning as well as deformation control, three new complexities are introduced. Firstly, as shown in Fig. 1.2, the measurement resolution highly depends on the density of the markers. More markers are necessary to gain finer details of an object, however, it is infeasible to put intensive number of markers on the object (e.g. human face). Therefore, the measurement resolution is usually lower than the requirement. Secondly, the hardware setup in these measurement systems is complex. To obtain a good 3D model, dense arrays of multiple professional machine vision cameras are required, which also make an accurate camera calibration a challenging task. Lastly, the parameter fitting step brings about a heavier computation burden. The whole process of getting the final results consumes several to dozens of hours [8].

1.2 Optical measurement techniques

Optical measurement techniques [7] are a well-known category of shape and deformation measurement techniques originated from experimental mechanics. They also aim at measuring the shape and deformation from real objects. They are,

however, superior to the computer-vision based methods in three aspects, which are summarized in Table 1.1. First, the optical measurement techniques focus on the measurement accuracy and precision. With the utilization of precision optics, a micro-scale or even a nano-scale measurement accuracy has been achieved. Furthermore, instead of using markers, the information of shape and deformation can be extracted from the patterns (e.g. fringe patterns or random speckle patterns) exerted on an object. Therefore, the measurement resolution is high. If necessary, a full-field result can be obtained with the resolution equivalent to the optical devices in use. Based on these two characteristics, optical measurement techniques have been extensively applied to the areas where accuracy is the main concern.

Table 1.1: Statistics of computation time in different stages for the seDIC and the paDIC.

	Computer-vision based methods	Optical measurement techniques
Precision	Millimeter [8]	Micro-scale[18], nano-scale [19], sub-pixel [20]
Resolution	Low, depend on the density of markers	High, full-field, no marker is needed
Application	Computer game, virtual reality, education	Ultra-precision fabrication, experimental mechanics, material science, etc..

Digital image correlation (DIC) and fringe pattern analysis (FPA) are two important categories of optical measurement techniques that are further investigated in this study. DIC is performed in white light, and only one or two cameras are needed for 2D in-plane [20] and 3D [21] measurement, respectively. A displacement or strain field is measured by the correlation analysis on the random speckle pattern painted or pasted onto an object. On the contrary, FTA uses regular fringe patterns or speckle patterns that are generated by the well-known interference phenomenon to carry the information of deformation. The shape of an object is then

obtained by phase extraction and phase unwrapping processes. In the past decade, both DIC and FPA have been well developed and successfully applied to various applications.

Figure 1.3: Patterns used to carry the information of deformation: (a) DIC uses random speckle patterns; (b) FPA uses regular fringe patterns.

1.3 Digital image correlation

Since first developed by Peters and Ranson for stress analysis [22], DIC has been extensively investigated and become one of the most popular non-contact, full-field surface displacement measurement techniques [20]. Successful applications of DIC have been found in various areas including experimental mechanics [23–25], biomechanics [26–28], and material analysis [29, 30]

As shown in Fig. 1.4, DIC computes the motion of each image pixel by comparing the differences between the reference (undeformed) image and the target

Figure 1.4: Terms used in DIC. ROI: region of interest; POI: point of interest.

(deformed) image. In the reference image, a region of interest (ROI) is defined and uniformly divided into small grids. Each point on the grid within the ROI, which is called point of interest (POI), is then processed by DIC. DIC tracks the displacement vector $\mathbf{d} = (u, v)$ of a grid point by matching its gray intensity value in the reference and the target images. However, tracking only one individual POI is not reliable, since many matches holding the same intensity value as the one being tracked can be found in the target image. Therefore, to establish uniqueness, each POI in the reference image is always enclosed by a $(2M+1) \times (2M+1)$ subset. The subset, which is centered at the POI $P(x_0, y_0)$, then serves as the basic matching unit in DIC.

Two common metrics to identify the optimal matching between the correspond-

ing subsets in the reference and the target images are the cross-correlation (CC) [31–35] and the sum of squared-difference (SSD) criterion [36–38]. As advocated [20] and then proved [39] by Pan et al., however, the plain CC and SSD criteria are very sensitive to lighting conditions of image acquisition systems (Fig. 1.3 (a)). Thus, the zero-normalized CC (ZNCC) and the zero-normalized SSD (ZNSSD), which are robust to light fluctuations, are suggested to be applied. In DIC, to achieve a high accuracy, the minimization process is often performed by an initial integer-pixel displacement estimation followed by a sub-pixel registration associated with a cubic interpolation [20].

1.3.1 Integer-pixel estimation

The advantage of utilizing a speckle pattern is its randomness: no two subsets are exactly the same, thus providing an easy way of obtaining an integer-pixel estimate for the deformations. If the deformation is small, a zero displacement could be directly used as initial estimates. Alternatively, the initial estimates can be obtained by manually selecting several corresponding points between the reference and target images [40]. However, an automatic approach has been widely used to compute more reliable initial integer-pixel estimates by performing a global search using the ZNCC criterion:

$$C_{ZNCC}(u, v) = \frac{\sum_i (R_i - R_m)(T_i - T_m)}{\sqrt{\sum_i (R_i - R_m)^2 \sum_i (T_i - T_m)^2}}, \quad (1.3)$$

where R_i and T_i are corresponding pixels in the reference and the target subsets, respectively; $R_m = \frac{1}{N} \sum_{i=0}^{N-1} R_i$ and $T_m = \frac{1}{N} \sum_{i=0}^{N-1} T_i$ refer to mean intensity values within the two subsets. The calculation of $C_{ZNCC}(u, v)$ can be simply carried out using a coarse-to-fine and nested search scheme in space domain[31]. Alternatively, it can be obtained from a fast Fourier transform based cross-correlation (FFT-CC) algorithm in frequency domain, since ZNCC in space domain is equivalent to a

point-wise multiplication in frequency domain [32], i.e

$$C_{ZNCC}(u, v) = FFT^{-1} \left(FFT^* \left[\frac{\sum_i (R_i - R_m)}{\sqrt{\sum_i (R_i - R_m)^2}} \right] \cdot FFT \left[\frac{\sum_i (T_i - T_m)}{\sqrt{\sum_i (T_i - T_m)^2}} \right] \right) \quad (1.4)$$

where FFT and FFT^{-1} denote the fast Fourier transform and inverse fast Fourier transform, respectively; and the superscript “*” represents the complex conjugate. The integer-pixel displacement $\mathbf{d} = (u, v)$ for each POI is then determined by searching the positive peak $C_{ZNCC}(u, v)$ coefficients within each subset.

1.3.2 Sub-pixel registration

An integer-pixel level displacement is not accurate enough in practical applications. It is thus always applied as the initial guess to a subsequent sub-pixel refinement algorithm. Various sub-pixel DIC algorithms, such as the coarse-to-fine search algorithm [22, 31], the spatial-gradient-based algorithm [40] as well as the Newton Raphson (NR) algorithm [41] have been found to be developed in the literature, among which NR has been proved to provide the most accurate and robust results [42].

1.3.2.1 Forward additive Newton Raphson algorithm

As shown in Fig. 1.4, suppose that an arbitrary point $Q(x_i, y_i)$ in the reference subset deforms to a point $Q'(x'_i, y'_i)$ in the target subset. The mapping from $Q(x_i, y_i)$ to $Q'(x'_i, y'_i)$ is always expressed by a first-order shape function [20] as

$$\begin{aligned} x'_i &= x_i + u + u_x \Delta x + u_y \Delta y \\ y'_i &= y_i + v + v_x \Delta x + v_y \Delta y \end{aligned} \quad (1.5)$$

where u_x , u_y , v_x and v_y represent the gradients of the displacement vector $\mathbf{d} = (u, v)$; and Δx and Δy are the local coordinates of point $Q(x_i, y_i)$ with respect to the POI $P(x_0, y_0)$. Equation. 1.5 can be rewritten as

$$Q' = P + \mathbf{W}(\xi; \mathbf{p}), \quad (1.6)$$

where $\mathbf{W}(\xi; \mathbf{p})$ is the warp function,

$$\mathbf{W}(\xi; \mathbf{p}) = \begin{bmatrix} 1 + u_x & u_y & u \\ v_x & 1 + v_y & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ 1 \end{bmatrix} \quad (1.7)$$

where $\xi = (\Delta x, \Delta y, 1)^T$ is the local coordinate of point Q within the reference subset, and $\mathbf{p} = (u, u_x, u_y, v, v_x, v_y)$ represents the vector of deformation parameters. The deformation parameter $\mathbf{p}$ can thus be determined by minimizing the nonlinear ZNSSD criterion,

$$C_{ZNSSD}(\mathbf{p}) = \sum_{\xi} \left\{ \frac{R(\mathbf{P}) - R_m}{\sqrt{\sum_{\xi} [R(\mathbf{P}) - R_m]^2}} - \frac{T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m}{\sqrt{\sum_{\xi} \{T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m\}^2}} \right\}^2 \quad (1.8)$$

Then $\mathbf{p}$ can be solved by the forward additive NR method as

$$\mathbf{p} = \mathbf{p}_0 - \mathbf{H}^{-1} \nabla C(\mathbf{p}_0), \quad (1.9)$$

where $\mathbf{p}_0$ is the initial guess estimated in Section 1.3.1 and $\mathbf{p}_0 = (u, 0, 0, v, 0, 0)$; $\nabla C(\mathbf{p}_0)$ and $\mathbf{H} = \nabla \nabla C(\mathbf{p}_0)$ are the gradient and Hessian of C_{ZNSSD} , respectively. Details of the Newton Raphson algorithm can be found in [41, 42].

1.3.2.2 Inverse compositional Gauss-Newton algorithm

NR has a very low computation efficiency, because the $\nabla C(\mathbf{p}_0)$ and $\mathbf{H}$ have to be updated every iteration. An improved NR was thus proposed by Vendroux and Knauss [43], which significantly reduced its computation complexity. They discovered that an approximated Hessian matrix, i.e $R(\mathbf{x}) \approx T[\mathbf{x} + \mathbf{W}(\xi; \mathbf{p})]$, could be able to produce a satisfying result when $\mathbf{p}$ is close to the real solution. Nonetheless, the repeat evaluation of $\mathbf{H}$ for the target image every iteration is still inefficient. In order to eliminate this redundancy, an inverse compositional Gauss-Newton (IC-GN) algorithm was proposed, which was proved to be equivalent to the forward additive Newton Raphson in terms of accuracy [44, 45] and was

employed in the RG-DIC [46]. Different from NR in which results of each iteration are added to the $\mathbf{p}$ to approach the real position in the target subset, IC-GN only adjusts the reference subset by updating an incremental deformation vector $\Delta\mathbf{p} = (\Delta u, \Delta u_x, \Delta u_y, \Delta v, \Delta v_x, \Delta v_y)$ instead of $\mathbf{p}$. The C_{ZNSSD} used in IC-GN is thus rewritten as

$$C_{ZNSSD}(\Delta\mathbf{p}) = \sum_{\xi} \left\{ \frac{R(\mathbf{P} + \mathbf{W}(\xi; \Delta\mathbf{p})) - R_m}{\sqrt{\sum_{\xi} [R(\mathbf{P} + \mathbf{W}(\xi; \Delta\mathbf{p})) - R_m]^2}} - \frac{T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m}{\sqrt{\sum_{\xi} (T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m)^2}} \right\}^2. \quad (1.10)$$

where the incremental warp function $\mathbf{W}(\xi; \Delta\mathbf{p})$ used to adjust the reference subset shape is

$$\mathbf{W}(\xi; \Delta\mathbf{p}) = \begin{bmatrix} 1 + \Delta u_x & \Delta u_y & \Delta u \\ \Delta v_x & 1 + \Delta v_y & \Delta v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ 1 \end{bmatrix}. \quad (1.11)$$

Equation 1.10 is a least squares problem with respect to $\Delta\mathbf{p}$ and can be solved by making $\frac{\partial C_{ZNSSD}}{\partial \Delta\mathbf{p}} = 0$. The solution for $\Delta\mathbf{p}$ is given as

$$\Delta\mathbf{p} = \mathbf{H}_{6 \times 6}^{-1} \sum_{\xi} \left\{ \left[\nabla R \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[\frac{\bar{R}}{\bar{T}} (T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m) - (R(\mathbf{P} + \xi) - R_m) \right] \right\}, \quad (1.12)$$

where $\bar{R} = \sqrt{\sum_{\xi} [R(\mathbf{P} + \mathbf{W}(\xi; \Delta\mathbf{p})) - R_m]^2}$, $\bar{T} = \sqrt{\sum_{\xi} \{T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m\}^2}$; ∇R is the gradient within the reference subset,

$$\nabla R = \left(\frac{\partial R(x, y)}{\partial x}, \frac{\partial R(x, y)}{\partial y} \right); \quad (1.13)$$

$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the Jacobin matrix of the warp function $\mathbf{W}(\xi; \mathbf{p})$,

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} 1 & \Delta x & \Delta y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta x & \Delta y \end{bmatrix}; \quad (1.14)$$

and $\mathbf{H}_{6 \times 6}^{-1}$ is the inverse of the 6×6 approximated Hessian matrix $\mathbf{H}$ [43],

$$\mathbf{H} = \sum_{\xi} \left\{ \left[\nabla R(\mathbf{P} + \xi) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[\nabla R(\mathbf{P} + \xi) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \right\}. \quad (1.15)$$

Every iteration, if the convergence criterion is not satisfied, $\mathbf{W}(\xi; \mathbf{p})$ is updated as

$$\mathbf{W}(\xi; \mathbf{p}) = \mathbf{W}(\xi; \mathbf{p})\mathbf{W}^{-1}(\xi; \Delta\mathbf{p}), \quad (1.16)$$

where $\mathbf{W}^{-1}(\xi; \Delta\mathbf{p})$ represent the inverse of the incremental warping function. As shown in Eqs. 1.14, 1.13, and 1.15, the Hessian matrix $\mathbf{H}$ only depends on the reference subset R so that it can be precomputed, thus the redundant update of $\mathbf{H}$ every iteration is eliminated in the IC-GN algorithm, which makes it superior to NR in terms of computation efficiency yet without compromising with accuracy.

1.3.2.3 Initial Gauss Transferring Schemes

A well-estimated initial guess is crucial for the convergence of the minimization procedure involved in the sub-pixel registration. Typically, there are two categories of initial guess transferring schemes, namely path-dependent and path-independent schemes.

Path-dependent scheme. In conventional DIC methods [40], it is believed that estimating the integer-pixel displacements for all the subsets is time-consuming, and this integer-pixel estimates are not adequate for fast convergence of the minimization process. Therefore, the initial guess is only estimated for the upper-left POI. Thereafter, based on the assumption of continuous deformation, the calculated result of the first POI is transferred as the initial guess to the next POI. These DIC methods which are based on an initial guess transferring scheme are thus named path-dependent methods. Nonetheless, the introduction of the path-dependence gives rise to another severe issue: the computation errors may propagate across subsets during the transfer. If initial guesses received from previous processed POIs are of low quality, the followings will very likely to give even worse results. This issue is especially serious if holes or discontinuous areas appear on the measured surface of a specimen, where an initial guess is always hard to be reliably estimated. In order to obtain reliable and robust initial guesses as well as improve the computation efficiency of DIC, Pan *et al* proposed a reliability-guided initial DIC (RG-DIC) method [46, 47]. In RG-DIC, the initial guess transfer is

guided by the zero-normalized cross-correlation (ZNCC) coefficients. The calculation begins with a carefully chosen seed POI, and the ZNCC coefficients of its four neighboring POIs are calculated. The POI with the highest ZNCC coefficient is then processed first and chosen as the new seed POI. As the calculation always moves along the most reliable path, the errors caused by discontinuous deformations as well as the propagation of errors are effectively reduced. Furthermore, the one-step direct transfer avoids the more complicated two-step method, added with the fact that an accurate initial guess does trigger a rapid convergence of the following sub-pixel registration procedure, the computation efficiency of RG-DIC is also increased significantly.

Path-independent scheme. The RG-DIC method though improves the computation efficiency of DIC by providing good initial estimates, its further performance boost is still highly obstructed by its path-dependence [48]. In addition, it fails to obtain a full-field deformation if the ROI contains large regions of discontinuity, since the boundary of these regions may propagate across the entire captured image. Indeed, powered by parallel computing [49–51], obtaining integer-pixel estimates for all the subsets is not time-consuming any more. Therefore, path-independent DIC methods accelerated by parallel computing [51, 52] are preferred in studying dynamic phenomena where a high processing speed is required. Nevertheless, a small deformation is implicitly assumed in path-independent DIC methods.

1.3.2.4 Interpolation

As observed from Eq. 1.12, intensity values need to be reconstructed by $T[\mathbf{x} + \mathbf{W}(\xi; \mathbf{p})]$ in a target subset, and a point calculated from $\mathbf{x} + \mathbf{W}(\xi; \mathbf{p})$ is very likely to fall at a sub-pixel location, as shown in Fig. 1.5, where a sub-pixel interpolation scheme has to be employed to evaluate the intensity values for the sub-pixel position $C(x, y)$. Various interpolation schemes, such as bilinear interpolation, bicubic interpolation, and bicubic B-spline interpolation, have been applied in the literature of the sub-pixel DIC methods. Though higher-order interpolation schemes

Figure 1.5: Schematic of the calculation of bicubic interpolation coefficients for a sub-pixel position $C(x, y)$.

provide better results, the bicubic interpolation scheme (Fig. 1.5) is always chosen because of its good trade-off between accuracy and computation efficiency. The bicubic interpolation is defined as

$$T(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 \alpha_{ij} x^i y^j, \quad (1.17)$$

where (x, y) is a sub-pixel location, and the 16 interpolation coefficients α_{ij} are determined by a grid of 4×4 integer pixels surrounding the sub-pixel location (the point $C(x, y)$ shown in Fig. 1.5). Therefore, the bicubic interpolation is very time-consuming if a large number of $\mathbf{x} + \mathbf{W}(\xi; \mathbf{p})$ refers to a sub-pixel location. Pan and Li then found that α_{ij} are invariant for any sub-pixel locations (shaded area in Fig. 1.5) surrounded by the same 4×4 integer pixels so that the α_{ij} 's for an entire image can be precomputed and formed up as a look-up table [53, 54], thus eliminating the redundant calculations. The combination of the IC-GN algorithm and the interpolation look-up table approach, which provides an excellent performance in terms of both accuracy and computation efficiency, has been successfully employed for sub-pixel registration in RG-DIC [46, 55] and the path-independent DIC (PiDIC) method [48].

1.3.3 Finite element-based global digital image correlation

The above mentioned DIC methods belong to the subset-based category. There is another category of DIC methods called Global DIC methods [56–58]. These methods minimize the SSD in Eq. 1.8 over the entire ROI at one time instead of any given subsets. The finite element method (FEM) is usually used to discretize the ROI. In global DIC methods, quadrilateral four-noded (Q4) elements are mostly used. Thus, the grid within the ROI (as shown in Fig. 1.4) becomes the finite elements, and the grid intersection points become their nodes. Global DIC methods track all the elements in the deformed image and obtain all the nodal displacements at the same time.

Figure 1.6: Global DIC using eight-noded (Q8) elements to discretize the boundary positions.

There are mainly two advantages of using global DIC methods. First, the most significant advantage of global DIC methods is that nodal results calculated using global DIC methods can be directly used to verify the FEM simulation obtained from mechanical identification processes [59]. Second, as shown in Fig. 1.6, the irregular finite elements can be easily used to handle boundaries or discontinuities

within the specimen. This process can even be automated by a self-adaptive algorithm [60]. However, the computation burden of global DIC methods is much heavier than those subset-based methods due to the global minimization process, which also restricts the possibility of acceleration using parallel computing.

1.4 Fringe pattern analysis and dynamic phase retrieval

Fringe patterns are the production of interference phenomena. Fringe pattern analysis, including denoising, phase retrieval, and phase unwrapping, is required to extract shapes and deformations that carried out by fringe patterns. While static fringe pattern analysis techniques have been extensively studied and successfully applied to various applications [61], dynamic extraction of phases from a sequence of fringe patterns is required to study the dynamic behaviors of an evolving field [62].

1.4.1 Fringe pattern sequence model

The phase distribution to be determined can be defined in terms of space and time as

$$\varphi(x, y; \tau) = \bar{\varphi}(x, y; \tau_0) + \Delta\varphi(x, y; \tau_0, \tau). \quad (1.18)$$

where $\varphi(x, y; \tau)$ and $\bar{\varphi}(x, y; \tau_0)$ are the current phase at time τ and the already calculated phase at time τ_0 , respectively; and $\Delta\varphi(x, y; \tau_0, \tau)$ is the phase change between the two time instants. The corresponding fringe pattern sequence is then defined as

$$\begin{aligned} f(x, y; \tau) &= a(x, y; \tau) + b(x, y; \tau)\cos[\varphi(x, y; \tau)] \\ &= a(x, y; \tau) + b(x, y; \tau)\cos[\bar{\varphi}(x, y; \tau_0) + \Delta\varphi(x, y; \tau_0, \tau)], \end{aligned} \quad (1.19)$$

where $(x, y; \tau) \in [0, N_x - 1] \times [0, N_y - 1] \times [0, K - 1]$ represent the spatial and temporal coordinates; $f(x, y; \tau)$, $a(x, y; \tau)$, and $b(x, y; \tau)$ represent the fringe pattern intensity, background, and amplitude, respectively. Figure 1.7 shows an example of

Figure 1.7: An example of dynamic phase retrieval for speckle sherography: (a)-(c) speckle correlation fringe patterns at three representative frames; (d)-(f) the corresponding retrieved wrapped phase maps.

dynamic phase retrieval of a sequence of speckle correlation fringe patterns. Most of the existing static phase retrieval methods, including phase shifting methods and transform-based methods, can be straightforwardly extended to dynamic cases.

1.4.2 Phase-shifting methods

Phase shifting [63, 64] is the most frequently used technique due to its simplicity and high accuracy. It can be applied to each single frame of $f(x, y; \tau)$. For example,

for $\tau = \tau_0$, a four-step phase shifting algorithm [64] performed as

$$\begin{aligned} f_0(x, y; \tau_0) &= a(x, y; \tau_0) + b(x, y; \tau_0) \cos [\varphi(x, y; \tau_0)], \\ f_1(x, y; \tau_0) &= a(x, y; \tau_0) + b(x, y; \tau_0) \cos \left[\varphi(x, y; \tau_0) + \frac{\pi}{2} \right], \\ f_2(x, y; \tau_0) &= a(x, y; \tau_0) + b(x, y; \tau_0) \cos [\varphi(x, y; \tau_0) + \pi], \\ f_3(x, y; \tau_0) &= a(x, y; \tau_0) + b(x, y; \tau_0) \cos \left[\varphi(x, y; \tau_0) + \frac{3\pi}{2} \right]. \end{aligned} \quad (1.20)$$

If constant background and amplitude are assumed, the phase can be retrieved as

$$\varphi(x, y; \tau_0) = \text{angle} \left[\frac{f_3(x, y; \tau_0) - f_1(x, y; \tau_0)}{f_0(x, y; \tau_0) - f_2(x, y; \tau_0)} \right]. \quad (1.21)$$

Doing phase shifting every frame is obviously very time-consuming and not suitable for dynamic processes. Therefore, a high speed temporal phase shifting method has been attempted [65], where phase-shifted specklegrams were shared by neighboring time instances. Spatial phase-shifting methods [66, 67] are good alternatives, in which phase-shifted fringe patterns are acquired simultaneously for a single shot at different locations. Both the temporal phase shifting and the spatial phase shifting methods, however, require complex and expensive optical setups.

1.4.3 Transform-based methods

Transform-based methods [68–75] are able to retrieve the $\varphi(x, y; t)$ from a single fringe pattern, however, this method requires either a temporal or a spatial carrier to separate the spectrum of the fringe pattern in the frequency domain. For example, at time $t = t_0$, a carrier fringe pattern can be written as

$$f(x, y; \tau_0) = a(x, y; \tau_0) + b(x, y; \tau_0) \cos [\varphi(x, y; \tau_0) + \omega_{cx}x + \omega_{cy}y], \quad (1.22)$$

where ω_{cx} and ω_{cy} are the known spatial carrier frequencies in x and y directions, respectively. Rewrite Eq. 1.22 based on the Euler's theorem as

$$\begin{aligned} f(x, y; \tau_0) &= a(x, y; \tau_0) \\ &+ c(x, y; \tau_0) \exp[j(\omega_{cx}x + \omega_{cy}y)] + c^*(x, y; \tau_0) \exp[-j(\omega_{cx}x + \omega_{cy}y)], \end{aligned} \quad (1.23)$$

where

$$c(x, y; \tau_0) = \frac{1}{2}b(x, y)\exp[j\varphi(x, y; \tau_0)], \quad (1.24)$$

and $*$ denotes the complex conjugate. The Fourier transform of Eq. 1.24 gives the spectrum of the $f(x, y; \tau_0)$ as

$$\begin{aligned} \mathfrak{F}[f](\xi_x, \xi_y; \tau_0) = \\ \mathfrak{F}[a](\xi_x, \xi_y; \tau_0) + \mathfrak{F}[c](\xi_x - \omega_{cx}, \xi_y - \omega_{cy}; \tau_0) + \mathfrak{F}[c]^*(\xi_x + \omega_{cx}, \xi_y + \omega_{cy}; \tau_0), \end{aligned} \quad (1.25)$$

where the $\mathfrak{F}[*]$ represents the Fourier transform. Since the added carrier always has a high frequency, the three terms $\mathfrak{F}[a](\xi_x, \xi_y; \tau_0)$, $\mathfrak{F}[c](\xi_x - \omega_{cx}, \xi_y - \omega_{cy}; \tau_0)$, and $\mathfrak{F}[c]^*(\xi_x + \omega_{cx}, \xi_y + \omega_{cy}; \tau_0)$ can be easily separated in the frequency domain. If a band-pass filter centered at $(\omega_{cx}, \omega_{cy})$ is applied to $\mathfrak{F}[f](\xi_x, \xi_y; \tau_0)$, only the $\mathfrak{F}[c](\xi_x - \omega_{cx}, \xi_y - \omega_{cy}; \tau_0)$ is kept after the filtering process. Therefore, $c(x, y; \tau_0)$ can be simply obtained by an inverse Fourier transform, and the phase is extracted as

$$\varphi(x, y; \tau_0) = \text{angle}[c(x, y; \tau_0)]. \quad (1.26)$$

The significant disadvantage of transform-based methods, however, is that the introduction of a temporal carrier limits the measurement range.

1.4.4 Reference-based methods

In reference-based methods [76–81], the phase of the first speckle pattern is determined by phase shifting methods [63, 64, 82] and then used as a reference. The phases of the rest fringe patterns are determined by estimating the phase changes between the current and the reference fringe patterns. These methods neither need a temporal carrier nor are required to perform phase shifting for each individual frame. Thus, they have the potential to be more widely used in studying high-speed dynamic behaviors. It is worth noting that, however, decorrelation and error accumulation are two severe problems which should be solved carefully to obtain correct and robust measurement results.

To solve these two problems, Li and Qian proposed a least-squares with three unknowns (LS3U) algorithm [62, 81], in which decorrelation was avoided by a reference updating scheme and error accumulation was eliminated by a windowed Fourier filtering (WFF) process. Moreover, the proposed method was performed in a pixelwise manner which can be accelerated by parallel computing and integrated into dynamic measurement applications.

1.5 Contributions and organization

1.5.1 Objective

As mentioned in Section 1.2, DIC and FPA, as two main categories of optical measurement techniques, have been extensively studied and well established. However, two issues still remain open for further investigation. First, as higher resolutions and higher measurement accuracy are pursued, the computation burden of DIC and FPA is becoming heavier, rendering them hardly be applicable to study dynamic phenomena or integrated into real-time systems. Second, decorrelation is a serious problem when the deformation between the initial and the deformed configurations of an object is too large to be tracked. Indeed, these two issues can be compensated by one solution: increasing the computation efficiency. Thus, the high-speed requirement in dynamic or real-time experiments can be fulfilled. The improvement of processing speed also provides a resolution of large deformation by providing the possibility to insert multiple intermediate frames between the initial and the end states such that the inter-frame deformation can be small enough to use existing algorithms without scarifying computation efficiency. Recently, parallel computing platforms composed of CPUs and graphics processing units (GPUs) have been extensively employed to accelerate optical measurement techniques due to their cost-effectiveness, easy programming interface, and great portability and scalability [83]. Therefore, in this thesis, a systematic study of powering DIC and FPA techniques using CPU and GPU parallel computing is investigated.

1.5.2 Contributions

Following the objectives described above, the contributions of this thesis are summarized as below.

- **Review of the CPU and GPU parallel computing applied to experimental mechanics and optical measurement techniques.** The CPU and GPU parallel computing techniques that have been successfully applied to optical measurement techniques, including digital image correlation (DIC), digital volume correlation (DVC), fringe pattern analysis (FPA), tomography, hyperspectral imaging (HSI), computer-generated holograms (CGH), and integrated imaging (II), are first reviewed [83]. We consider this review as a contribution for the reason that it is the first review paper regarding CPU and GPU parallel computing in the areas of experimental mechanics and optical measurement. We intend to raise the awareness of researchers in these areas about the importance of CPU and GPU parallel computing. Moreover, four parallel patterns that can be commonly exploited from optical measurement techniques are extracted and explained, which serve as the foundation of the following works, and can even be used as a guidance for accelerating all the similar problems that may be encountered in future applications.
- **Parallel DIC algorithm.** A GPU powered parallel DIC (paDIC) algorithm [51] is proposed, which employs a path-independent initial guess transferring scheme. In paDIC, both the integer-pixel displacement estimation and sub-pixel registration are calculated on the GPU. A 57.5+ times speedup has been achieved compared with the sequential implementation of same algorithm [48]. It is the fastest DIC algorithm using the 1st order (which is more accurate than the 0th order) shape function published in the literature yet without compromising with accuracy.

- **Parallel DVC algorithm.** The idea of the paDIC algorithm is extended to 3D DVC to study the internal volumetric deformation of an object, and a parallel DVC (paDVC) algorithm is proposed. DVC is much more complex than DVC due to the introduction of an extra dimension. Thus, to efficiently process the large amount of 3D data, a batch processing scheme is employed. The proposed paDVC is 23.3 times and 3.7 times faster than its optimized CPU sequential and multi-core implementations, respectively. It is the fastest pure GPU-based DVC method published so far.
- **Hybrid CPU-GPU real-time DIC system framework.** Based on paDIC and paDVC, a real-time DIC system framework, which combines the strength of both the CPU and the GPU is proposed [84]. While the main logics, such as graphical user interface (GUI), data acquisition, processing, and display are pipelined on the CPU side, the computationally intensive tasks are offloaded to the GPU side. With the flexibly designed three variations of the framework, a real-time frame rate ranging from 30fps (frames per second) to 130fps has been achieved. Also, a reference updating scheme is proposed to eliminate the decorrelation that may occur when the deformation becomes large.
- **Parallel reference-based real-time dynamic phase retrieval algorithm.** A real-time reference-based dynamic phase retrieval algorithm, called G-LS3U is proposed [85]. Different parallel computing strategies are applied to both the least-squares fitting and the WFF processes, achieving an extremely high processing rate at 131+fps. The G-LS3U algorithm is the fastest reference-based phase retrieval algorithm heretofore.

1.5.3 Organization

The rest of this report is organized as follows. Chapter 2 reviews the parallel computing techniques applied to experimental mechanics and optical measurement techniques in the past 5 years. The details of paDIC and paDVC are explained in

Chapter 3 and Chapter 4. Chapter 5 illustrates some thoughts on designing and implementing a real-time DIC system. Chapter 6 explains the G-LS3U algorithm. Finally, the conclusion and future work are summarized in Chapter 7.

Chapter 2

Literature review: CPU and GPU parallel computing in experimental mechanics and optical measurement

As mentioned in Section 1.2, because of the advantages of non-destructiveness, high sensitivity, and high accuracy, optical methods have been successfully proposed and applied to the measurement of various important physical quantities such as displacement, strain, stiffness, and density in the areas of experimental mechanics (EM) and optical measurement (OM) [86]. However, as the development of modern optical devices and in pursuit of higher measurement resolution, higher data acquisition and processing rates are required to process the resulting bulky data. Even worse, more complicated algorithms [36, 62] are being pursued to obtain higher accuracy and robustness, which adds additional computation burdens.

To accelerate data processing, two effective approaches have been employed: by smart implementation of an algorithm [53, 87] and by parallel computing [88]. In one previous research work, in 2011, Gao and Kemao [89] reviewed and compared various parallel computing hardware employed to boost the computation efficiency of optical methods. Basic parallel computing principles, such as data/task parallelism and pipelined parallelism together with different parallel computing devices,

including computer clusters, field-programmable gate arrays (FPGAs), pipelined processors, GPUs, etc. were explained. It was concluded that hardware acceleration is an effective approach for consideration when time-consuming processes required speed improvement or real-time performance was pursued in commercial applications. Specifically, it was mentioned that, from year of 2008, the GPU tends to be more widely applied than other parallel hardware, since efforts put into GPU programming have been significantly reduced, thanks to the invention of the Computer Unified Device Architecture (or CUDA) [90].

GPU technology has already become the mainstream hardware accelerator in the past five years since 2012. It continues to explode in the areas of EM and OM. However, the CPU is still an important parallel hardware: (i) the GPU cannot work without an association with the CPU in the current computer architecture; (ii) multi-core CPUs are still convenient accelerators for parallel applications; (iii) the CPU and the GPU have their own strength and weakness in parallel computing due to their different design philosophy. Thus, to tackle the optimal acceleration, hybrid parallel platforms exploiting and combining the strength of both the CPU and the GPU have been successfully applied to high-performance and real-time applications; (iv) super computers, Titan as an example (contains 299K AMD x86 cores and 18.6K NVIDIA GPUs) [91], which is one of the world's top 3 super-computers, always feature of multiple nodes of CPUs and GPUs based on their excellent scalability and portability.

Thus, in this chapter, the CPU and GPU parallel computing techniques applied to EM and OM are reviewed. The architectures of the CPU and the GPU as well as the CUDA programming model are briefly introduced first, followed by the explanation of the four common parallel patterns that can be simply identified and efficiently parallelized in a series of existing applications. Thereafter, the CPU and GPU parallel computing applied to the areas of DIC/DVC, FPA, tomography, HSI, CGH, and II are reviewed. The specific common parallel patterns employed in each reviewed work are explained to make them easier to be reused in future applications. Note that, to keep this thesis concise, DIC/DVC and FPA techniques,

which are the main focuses, are reviewed in more detail than the others. The discussion regarding the current state and further trend of applying the CPU and GPU parallel computing in EM and OM are found at the end of this chapter.

2.1 Parallel computing on multi-core CPUs and GPUs

As mentioned above, the CPU and the GPU have different design philosophy, which make them superior in specific parallel tasks. In this section, the architectures of the CPU and the GPU are first introduced. Based on their architectures, certain existing problems in optical methods can be abstracted as four common parallel patterns which can be effectively parallelized. These parallel patterns are analyzed and are followed by an introduction of several existing well-optimized libraries that can be employed directly to facilitate their implementations.

2.1.1 CPU and GPU architectures

The architectures of the CPU and the GPU are schematically shown in Figs. 2.1 and 2.2. According to the Flynn's taxonomy [92, 93], modern multi-core CPUs are designed with the multiple instructions, multiple data (MIMD) architecture, which is superior in task parallelism, with each core scheduled different tasks by the operation system (OS). As shown in Fig. 2.1, typically, a modern CPU contains 4-32 cores. Each core is optimized for serial code execution by the employment of large and multi-level caches, low-latency arithmetic and logic units (ALUs), branch prediction units, etc. Such optimizations consume a large area of a CPU chip, thus limiting its total number of cores.

Different from the CPU, the GPU is designed with the single instruction, multiple data (SIMD) architecture, which makes it superior in data parallelism. As shown in Fig. 2.2, the basic operation unit of the GPU is called streaming multiprocessor (SM), each of which consists a number of streaming processors (SPs) and a small cache memory. The NVIDIA's GTX 1080 card, as an example, has

Figure 2.1: Schematically illustration of the architecture of the CPU. Adapted with permission from [83], Opt Lasers Eng 0143-8166 (2017).

20 SMs, each contains 128 SPs, thus, totally there are $20 \times 128 = 2560$ SPs. The NVIDIA GPUs employ an improved SIMD design named SIMT (single instruction, multiple threads), which allows a portion of the GPU threads to repeat the same instruction, while others either remain idle or are re-targeted to execute other instructions. With this design, the GPU will get stuck when branches and loops present. But it gives room to a GPU chip to contain and spawn massively many such threads, increasing the total throughput. It is worth mentioning that the GPU can only approach its peak throughput when the size of a problem is large enough to fully utilize its computation resources.

Based on their design philosophies, the practical usage of the CPU and the GPU in parallel computing can be summarized as below. The CPU is preferred to the GPU in two aspects: (i) CPU threads can be assigned different tasks asynchronously. Every thread maintains its own execution logic, while the consumption of inter-thread communications is cheap and can be simply carried out by shared concurrent buffers. (ii) the CPU is faster than the GPU on small-scale data-parallel problems, which cannot fully utilize the GPU's bandwidth. The GPU, on the contrary, is superior to the CPU when the problem sizes are large and the involved instructions are simple. In such cases, even a low-end GPU is faster than a CPU by spawning and running tens of thousands of threads simultaneously. The more ideal usage of the CPU and the GPU is to unify their strengths in accelerating

Figure 2.2: Schematically illustration of the architecture of the GPU. Adapted with permission from ref [83], Opt Lasers Eng 0143-8166 (2017).

complicated algorithms, where the CPU is responsible for the main logic of an algorithm and the computationally intensive tasks can be off-loaded to the GPU [94]. This hybrid CPU-GPU technique has been widely used in EM and OM applications, in which CPU threads are responsible for camera control, data acquisition, data loading, processing, and storing, etc. The data processing thread is always responsible for scheduling and launching GPU routines.

2.1.2 Compute unified development architecture (CUDA)

Initially, the GPU is designed to process computationally intensive tasks for 3D graphics applications such as texturing a large set of polygons, calculating shading and lighting, and rendering the resulting image on screen. With the development of programmable shaders, the fixed GPU rendering pipeline can be manipulated

to calculate different effects. The GPU thus began to be used as a general-purpose parallel computing device, and the so-called general-purpose GPU (GPGPU) programming based on shading languages emerged.

Nonetheless, GPGPU programming at that time was very complex, since data in a shader should be treated as a set of 3D points that represented polygon meshes, but a lot of datasets could not be represented in this way. Even worse, shaders were usually tricky to debug, making it very difficult to obtain the expected results without a deep knowledge in computer graphics.

To make the GPGPU a real programmable device as the CPU, which has been well developed, many programming primitives have been proposed, for instance, Cg, OpenCL, and CUDA, or Compute Unified Device Architecture. Among them, CUDA, which was first released by NVIDIA in 2007, has become the mainstream by providing the easiest-to-use programming interface.

Figure 2.3: Overview of the CUDA programming model and its mapping to the NVIDIA GPU hardware. SM: Streaming Multiprocessors.

Figure 2.3 shows the schematic of the CUDA programming model and its mapping to the NVIDIA GPU hardware architecture. The host code can spawn CUDA kernels and transfer data to or from per grid global and constant memories on the device side. Within a CUDA kernel, a parallel problem is first split into coarse sub-problems that are allocated to multiple CUDA blocks, each sub-problem is then

further divided into finer pieces that can be processed by all the CUDA threads within each block. For hardware, an NVIDIA GPU is composed of several streaming multiprocessors (SMs), each of which contains a lot of CUDA cores (e.g. an NVIDIA GTX 1080 GPU has 8 SMs, each SM has 320 CUDA cores). To execute the kernel on the GPU, CUDA blocks are mapped to SMs and scheduled to different CUDA cores in runtime. There are mainly four types of memories residing in the CUDA programming model. Sorted by their latency in ascending order, they are registers, shared memory, constant memory, and global memory.

Registers are not exactly memory. They are bytes stored in global memory but automatically cached in the on-chip L1 cache so that it can be accessed. The size of each register is 4 bytes, and each thread can contain up to 63 registers. Individual variables or arrays defined with constant numbers of elements can be automatically saved in registers. However, if the per-thread source code exceeds the register limit, it will be offloaded to global memory.

Shared memory, as its name indicates, is shared by the threads within a CUDA block. It shares the same on-chip memory space with L1 cache, but it is programmable. The size of the overall on-chip memory is limited to up to 64KB for each SM on the GPU. Therefore, the size of shared memory has a direct influence on how many CUDA blocks can be scheduled to one SM on the GPU. The extreme case is that if the shared memory is exhausted in every CUDA block, only one block can be scheduled to this SM at a time.

Constant memory is a 64KB read-only memory space. Its content cannot be modified after it is defined at compile time. Since constant memory generally has lower latency than global memory, it is often used to store a small amount of data that will not change over the course of a kernel execution.

Global memory has the highest latency, but it is the largest and it is the memory to exchange data with the host. Due to its high latency, in practice, it is always recommended to minimize the number of host and device memory transfer. Also, pinned host memory (Section 6.4.1) is a common strategy that can be used to reduce the latency of global memory. The CPU data allocations are unpinned by

default. The GPU cannot access data directly from unpinned memory, so when a data transfer from unpinned memory to the GPU is invoked, the CUDA driver must allocate a temporary pinned memory first. This temporary pinned memory allocation can be avoided by pre-allocating the CPU memory as pinned memory.

2.2 Common parallel patterns

There are four parallel patterns that can be commonly extracted and parallelized in EM and OM applications: the pixelwise pattern, the tiling pattern, the divide-and-conquer pattern, and the rendering and interpolation pattern. Parallelization of these operations is simply to partition the work into small blocks and then allocate them to multiple processors or threads.

2.2.1 Pixelwise pattern

Pixelwise operations are commonly encountered in image processing in EM and OM applications and they are the easiest to be parallelized. The serial implementation of pixelwise operations is merely loops with no inter-loop dependencies. Parallelization of these operations is simply partitioning the work into small blocks and then allocate them to multiple processors or threads.

For the CPU, this is what OpenMP [95] and Intels TBB [96] are doing typically. When the code execution reaches the pixelwise loop part, N threads are spawned to perform it in parallel. However, N cannot be too large, otherwise the OS must do many context switching that lead to bad performance. Usually, N is fixed at the number of physical cores of a CPU or double if hyperthreading [97] is supported. Thus, the CPU can only achieve the so-called coarse-grained level parallelism. On the contrary, the GPU benefits from the fine-grained parallelism. As mentioned in Section 2.1.2, the GPU achieves its peak performance when its resources are fully occupied. Therefore, N on the GPU side is always configured as the multiple of the number of blocks one SM supports multiplied with the number of SMs to maximize the occupancy. Moreover, this configuration guarantees a good scalability when

new GPUs are released. A little more complicated situation is often found that these pixelwise loops are within other outer loops, which is simply handled by only parallelizing the outer loops on a CPU since it has a very limited number of cores. On the GPU side, one may consider to flatten the loops, since applications in EM and OM always deal with 2D images or 3D volumetric data, all of which can certainly be transformed into a 1D sequence with almost no overhead.

2.2.2 Tiling pattern

Tiling [98] is a typical parallel pattern for the GPU that uses the GPU's fast shared memory. It deals with calculations at one position that should include its neighbors. These neighboring operations are very common in EM and OM applications, such as convolution, gradients calculation, LUT (look-up table) generation [99], windowed or local operation, etc. Like the pixelwise pattern, data is first split into small tiles (or blocks), however, each tile now must also include the additional neighboring positions. To handle the boundaries within these tiles, every two consecutive tiles should be overlapped and the size of the overlapping regions is usually set to the number of neighbors involved in the operations to keep the maximum number of threads working on each tile busy.

2.2.3 Divide-and-conquer pattern

The divide-and-conquer pattern is also a pattern that breaks down a large problem into smaller sub-problems, each of which is then conquered. After all sub-problems are processed, the overall large problem is solved. Typical divide-and-conquer algorithms found in EM and OM applications include the fast Fourier transform (FFT) [100] and the group of reduction algorithms [101]. However, divide-and-conquer algorithms always include recursive processes, which should be carefully dealt with. Although GPUs with post-Fermi architectures and all CPUs support recursion, the stack overflow problem will happen if the call depth of the recursion is too deep. Fortunately, as every recursive problem can be converted to an iterative one, which may then be solved as the pixelwise pattern, it is recommended to perform this conversion first before parallelizing divide-and-conquer algorithms.

2.2.4 Rendering and interpolation pattern

GPUs are born-good renderers in computer graphics. Rendering is a series of processes that create a 2D display of a 3D scene following a graphics rendering pipeline, e.g. the OpenGL graphics pipeline [102]. In EM and OM applications, it is always required to visualize calculated results, such as displacement and strain fields. If these quantities are obtained on the GPU, they can be directly rendered and displayed on screen without moving them back to the CPU side.

Interpolation is another important operation in EM and OM, especially when a sub-pixel accuracy is required. It can be performed extremely fast on the GPU's texture memory, since linear interpolations are inherently hard-wired on the GPU. However, single linear interpolations always fail to provide the required accuracy. D. Ruijters et al as well as F. Champagnat and Y. Le Sant proposed efficient cubic interpolation methods on the GPU texture memory [103–105]. The main idea was that the cubic B-spline interpolation equation can be decomposed into several linear interpolations which could still benefit from the texture memory cache. Zhang et al and Wang et al also proposed a LUT approach to accelerate the bi-cubic and tri-cubic interpolations without using the GPU's texture memory [51, 99]. Thus, interpolation is recommended to be performed on the GPU since it possesses both high accuracy and speed.

2.3 Existing parallel computing libraries

Except for some ad hoc parallel problems, it is always recommended to refer to existing, fine-tuned, and high-performance parallel computing libraries before re-implementing from scratch. For the CPU, Intel Math Kernel Library (MKL) [106] provides fast and robust parallel implementations of BLAS and LAPACK routines; FFTW [107] is one of the best CPU FFT algorithm libraries. Their GPU counterparts are cuBLAS [108], cuSolver [109] and cuFFT [110] libraries. Thrust [111] is a CUDA library that provides many general routines for the optimal implementations of the common parallel patterns mentioned above together with a C++ Standard

Template Library (STL) like interface, making it much easier to parallelize existing serial codes. OpenCL [112], as an open-source alternative to CUDA, can be used on almost any brand GPUs and CPUs with SSE3 support. Nonetheless, the compilation of OpenCL code is complicated, and several convenient built-in functions of CUDA should be manually implemented in OpenCL. In practice, the proper usage of these libraries can enormously facilitate the development time and the performance is always better than one's own implementations.

2.4 Applications in experimental mechanics and optical measurement

In this section, the CPU and GPU parallel computing techniques applied to DIC&DVC, FPA, tomography, HSI, CGH, and II, are reviewed.

2.4.1 Digital image correlation

DIC is a simple yet effective technique for the measurement of surface displacement or strain under white light [20]. DVC is the 3D extension of DIC, which measures the 3D internal volumetric deformation of an object [113]. As mentioned in Section 1.3, both the subset-based local and the FEM-based global DIC&DVC methods are based on the correlation analysis. This process has very heavy computation loading [114] that prohibits DIC&DVC from high-speed or even real-time applications. Therefore, many CPU and GPU parallel computing strategies have been applied to accelerate both of these methods.

2.4.1.1 Subset-based local DIC&DVC algorithms

Integer-pixel accuracy methods. Integer-pixel level displacement is mainly calculated using theZNCC criterion (Eq. 1.3). GPU-accelerated integer-pixel level DIC method has been proposed by Gembris et al. [115], and applied in cracking analysis [116], biomechanics [49], and optical coherent tomography (OCT) [50]. The ZNCC can be either parallelized straightforwardly in space domain [50,

117] or in frequency domain using the fast Fourier transform (FFT) ZNCC (FFT-CC) algorithm [49, 116]. The FFT-CC algorithm takes the advantage that cross-correlation calculations in the frequency domain become pointwise complex number multiplications, which well fit the pixelwise pattern. Moreover, the cuFFT library can be directly used to implement the required forward and inverse FFTs (iFFTs). Therefore, the FFT-CC algorithm is more suitable for the GPU.

Sub-pixel accuracy, path-dependent methods. Integer-pixel level accuracy is always not enough for precision DIC measurement. A sub-pixel accuracy can be achieved by minimizing ZNSSD (Eq. 1.10) together with an interpolation process, however, it brings additional computation burdens. In path-dependent DIC methods, based on the continuous deformation, to obtain a fast convergence of the minimization process, an initial guess transferring scheme and a 1st or 2nd order shape function are commonly employed to minimize ZNSSD. Nonetheless, this path-dependence puts a barrier to parallelization. In the literature, only the CPU has been used to accelerated path-dependent DIC algorithms. Shao et al. proposed a multi-core implementation of the seed POI-based inverse compositional Gauss-Newton (IC-GN) algorithm [117]. The algorithm starts at the seed POI. The uncalculated neighbors of the seed POI are allocated to and calculated simultaneously by different CPU threads. This process repeats until all the POIs within an ROI have been processed. They then applied their method to realize a 3D real-time human pulse monitoring system running at a frame rate of 10 fps [21]. Instead of starting from only one seed POI, Wu et al. proposed a multi-point tracking scheme to parallelize their particle swarm optimization (PSO) and block-based gradient descent search (BBGDS) algorithms to realize their real-time DIC algorithm [118]. A 60 fps processing speed was claimed for a 4-POI strain measurement experiment. Pan et al. provided the multi-core implementation of their proposed accurate and robust RG-DIC method [119], achieving up to $7\times$ speedup over its serial implementation. This multi-core RG-DIC was then integrated into the well-known MATLAB-based DIC software called Ncorr [55].

Sub-pixel accuracy, path-independent methods. For further acceleration, path-independent DIC methods powered by GPUs emerged. Jandjsek et al.

first proposed a path-independent DVC algorithm based on the normalized CC (NCC) algorithm for integer-pixel displacement estimation and the Lucas-Kanade algorithm [45, 120] for sub-pixel registration. The parallel implementation of the proposed algorithm was foreseen yet not implemented [121]. Valle et al. proposed an H^k -DIC algorithm for cracks measurement, which was the first time to calculate each subset independently on the GPU. A $33\times$ speedup compared with its serial counterpart was achieved [122]. Likewise, a fast iterative DVC (FE-DVC) algorithm [123] which parallelized the weighted FFT-CC on GPU was proposed. It was claimed to be $5.5\times$ faster than its CPU counterpart.

Bar-Kochba *et al.* [124] proposed a DVC method that iteratively compares the reference volume image and the target volume image through FFT-CC algorithm and then adjusts the two images until the displacement increment obtained in iteration reaches a sufficiently small value. In their implementation, the FFT-CC computation at POIs can be performed in parallel on GPU or sequentially on CPU. The GPU implementation gained a speed improvement of about $5.5\times$ over its CPU counterpart. Gates et al. proposed an accurate DVC method which combined the FFT-CC with the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm for sub-pixel registration. They provided both the CPU multi-core [125] and the hybrid CPU-GPU [126] implementations of their method. In the latter implementation, CPUs were used to execute the BFGS iterations while the calculation of the NCCs as well as their gradients were off-loaded to GPUs. Tested on 3 GPUs and a 12-core CPU, it was $8\times$ faster than running the same algorithm on a single CPU.

Recently, Besnerais et al. proposed an extremely fast DIC method named FOLKI-D [52] for dense 2D and 3D displacement field estimation based on their previous fast particle image velocimetry (PIV) computation [127]. The GPU is responsible for the pixelwise operations such as solving the linear systems and sub-pixel bi-cubic spline interpolation on the GPU's texture memory [104]. With a 0^{th} order shape function and reducing the redundant calculations at the overlapped regions among subsets, FOLKI-D achieved a computation speed of 2.5×10^6 for a 75% overlap with 2D DIC.

2.4.1.2 FEM-based global DIC&DVC algorithms

As explained in Section 1.3.3, due to the global updating scheme, FEM-based DIC&DVC algorithms cannot be well parallelized. Thus, the GPU is mainly used for matrix construction and solving large linear systems in the kinematic equation as

$$\mathbf{M}\mathbf{A} = \mathbf{X}. \quad (2.1)$$

Leclerc et al. parallelized their integrated DIC (I-DIC) algorithm [59], in which one finite element was allocated to one CUDA block, and each thread within the block was responsible for one pixel in the element. In addition, the GPU's texture memory was used to construct the stiffness matrix $\mathbf{M}$ for its fast cache and interpolation capabilities. It was reported that only 1 second was required for megabyte images using the parallel I-DIC algorithm. The GPU was also employed by them to solve the linear system containing millions of degrees of freedom (DOFs) of the voxel-based global DVC method [128], and to construct the matrix $\mathbf{N}$ and the vector $\mathbf{n}$ of the projection-based (P-DVC) method [129]. A domain coupling method for global DIC algorithms was proposed [130] and parallelized [131] by Robin and Jean-Charles.

2.4.2 Fringe pattern analysis

Section 1.2 mentioned that, instead of using random speckle patterns as in DIC&DVC, fringe pattern analysis (FPA) relies on regular fringe patterns which are generated by the well-known interference phenomenon. In FPA, phase retrieval is the most essential stages for the extraction of required physical quantities. One main category of FPA methods is the fringe projection profilometry, in which fringe patterns are projected onto an object surface. The deformation is then measured by analyzing the deformed fringe patterns captured by digital cameras [62].

2.4.2.1 Fringe projection profilometry

Fringe projection profilometry (FPP) basically includes five stages: fringe pattern generation, data acquisition, phase extraction and denoising, phase-to-coordinate

conversion for profile reconstruction, and profile visualization. The acceleration has been implemented for either a certain stage or the whole process. Karpinsky et al. has proposed three research works on using GPU to realize real-time 3D shape measurement based on FPP [132–134]. In the first paper [132], the GPU was mainly used for powering the pixelwise phase-to-coordinate conversion, which enabled the reconstructed 3D data residing in the GPU memory to be rendered immediately by using the OpenGL graphics pipeline without transferring them back to the CPU side. Moreover, by substituting the sinusoidal fringe patterns with the binary structured patterns generated by the defocussing technique in the three-step phase-shifting algorithm, a 30 fps was achieved by the proposed system at a resolution of 300,000 points per frame. This idea was later integrated into a 3D range geometry video compression method [133], in which the pixelwise operations such as the phase calculation, phase filtering, and depth map and normal map calculations were performed on the GPU after obtaining decoded frames from the CPU. The phase encoding and decoding processes reached 17 fps and 25 fps on a modern laptop’s GPU, respectively.

As mentioned in Section 2.1.2, data transfer between the host (CPU) and the device (GPU) using the PCI-E bus is one of the major speed barriers in GPU programming. A pure GPU-based high-resolution, real-time 3D shape measurement system [134] was thus proposed to compensate for this drawback. In addition to the parallel computing strategies in [132, 133], the captured fringe patterns were directly loaded to the GPU’s global memory, thanks to the USB 3.0 and the direct memory access (DMA) technologies. The CPU was thus idle and was used to perform the 3D registration and feature extraction in parallel with the GPU. Multi-pass and off-screen rendering were also employed to further boost the overall performance. With all these optimizations, the proposed system could run at 30 fps with 480,000 measurement points per frame.

Pipelined systems fetching the computation strengths of both CPUs and GPUs have also been attempted. Nguyen et al. proposed a pipelined 3D imaging and measurement system [135], in which multiple CPU threads were responsible for

the six sub-tasks including fringe projection, image acquisition, etc. Each thread then off-loaded its computationally intensive tasks to the corresponding CUDA kernels. The results were directly rendered and displayed using OpenGL thanks to the CUDA and OpenGL interoperability [136]. The system achieved real-time frame rates of 45 fps and 22.5 fps for the measurement accuracies of 0.04% and 0.01%, respectively. A similar idea was introduced by Van Der Jeught in realizing a real-time microscopic profilometry system [137], in which data acquisition and processing were scheduled to two CPU threads and the entire computation was performed on the GPU. It was claimed to generate 30 microscopic height maps per second using the four-step phase shifting algorithm, and a potential speedup could be expected if the three-step phase shifting algorithm was used [138].

2.4.2.2 Other fringe pattern analysis techniques

The CPU and GPU parallel computing has also been used in windowed Fourier transform (WFT), phase quality map generation, phase denoising, and phase extraction.

Zhao and Kemao examined the optimized CPU multi-core implementation of the WFT algorithm [139]. Different performances of implementing the same algorithm using different libraries and compilers were compared, in which it was proved that the OpenMP threading library with the Intel's C++ compiler achieved the best speed performance.

Zhong et al. proposed a novel phase quality map generation algorithm and implemented it on the GPU [140]. The quality values of two arbitrary pixels were calculated by the local wrapped phases around the interrogating pixel, which perfectly fitted the tiling pattern. The proposed GPU-based implementation achieved a $75.7\times$ speedup compared with its single-core CPU counterpart. A CUDA optimized wrapped non-local means (WNLN) algorithm was proposed by Zimmer and Ghuman was applied for interferometric phase denoising [141]. Two CUDA kernels were involved in their implementation: (i) a optimized convolution kernel

which was parallelized by the tiling pattern; (ii) a weighted maximum likelihood estimation (WMLE) kernel which mainly benefited from the parallel reduction.

Zhu et al. parallelized the famous advanced iterative algorithm (AIA) [82] on the GPU for real-time phase extraction [142], where the GPU was responsible for the pixel-by-pixel operations while the CPU was used for the frame-by-frame iterations. In the proposed method, for a better trade-off between speed and accuracy, only the central 120×120 pixels of a 1024×1024 frame was considered in solving the frame-by-frame linear systems, eliminating 98% iterations on the CPU. It is worth noting that, however, as the summation among pixels in each frame can be parallelized by the parallel reduction, and even higher speed is expected by using the GPU. Even more, the summation can be performed on the entire image without compromising accuracy. Backoach et al. proposed a CUDA-accelerated fast phase extraction algorithm for off-axis holographic interference patterns [143]. The operations of extracting wrapped phases from off-axis holograms and phase unwrapping were parallelized on the GPU. Although the wrapped phase extraction process utilized FFT and could be directly accelerated by cuFFT[110], a path-independent, discrete cosine transform (DCT) based unweighted least squares (UWLS) algorithm was employed instead to provide more parallelism for the GPU. Besides, the solution of the Poisson's equation was parallelized with the pixelwise pattern. Thus, the entire phase unwrapping process could run at 40.7 fps for 1 mega pixel holograms. The same idea was then improved and extended to process 3D tomographic phase microscopy (TPM) which yielded a $64 \times 64 \times 64$ output under video-rate [144].

2.4.3 Tomography

Tomography is a 3D imaging technique using a penetrating wave to capture objects' internal structures. The CPU and GPU parallel computing has been widely adopted in optical coherence tomography (OCT) for image processing and 3D reconstruction processes. Other parallel tomographic applications include optical diffraction tomography (ODT), diffuse optical tomography (DOT), and photoacoustic tomography (PT).

2.4.3.1 Optical coherence tomography

OCT uses low-coherent interferometry to perform non-invasive imaging of shallow internal micro-structures. There are two categories of OCT methods: spectral-domain OCT (SD-OCT) and swept-source OCT (SS-OCT), both of which contain intensity image processing and 3D reconstruction processes, which have been well accelerated by the CPU and GPU parallel computing. Specifically, to overlap data acquisition and processing, the hybrid pipelined CPU-GPU architecture was widely adopted.

Spectral-domain OCT systems. An open-source SD-OCT system accelerated by dual GPUs was proposed by Li et al., in which the data-intensive tasks such as DC removal, resampling, dispersion compensation, FFTs, and logarithm were off-loaded to the two GPUs [145]. A $3\times$ speedup was then achieved by using CUDA streams to overlap data transfer and processing [146]. Based on [145, 146], a parallel wavefront sensorless adaptive optics (WSAO) SD-OCT method was applied to *in vivo* the imaging of small animal retinas [147] and human photoreceptor mosaics [148].

Wang et al. realized a real-time multi-functional SD-OCT system combining intensity OCT, Doppler OCT (DOCT), and polarization-sensitive OCT (PS-OCT) using the hybrid pipelined CPU-GPU architecture [149], in which five CPU threads were responsible for the synchronization of the cameras, the galvanometer-mounted-mirrors and the polarization modulator, data acquisition, and controlling the GPU, while the GPU performed intensity calculation, phase retardation, flow calculation, and visualization. Similarly, Huang et al. proposed a real-time 3D and 4D Fourier domain phase-resolved DOCT (PS-DOCT) and accelerated it using one CPU and dual GPUs [150]. The hybrid architecture was then employed to realize a motion-compensated hand-held common-path SD-OCT probe [151].

To solve the problems of limited depth range and low image quality caused by the dispersion of complex conjugate artifacts in SD-OCT, an iterative dispersion encoded full-range (DEFRR) algorithm was proposed. However, DEFRR suffered a low computation efficiency because of the two Fourier transforms within each

DEFR iteration. To solve this issue, Wang et al. accelerated DEFR by parallelizing the FFTs and iFFTs on the GPU [152]. Cho et al. proposed a high-speed SD-OCT system and applied it to *in vivo* human eye imaging, in which the GPU was used to parallelize the K-domain linearization, FFT, and the log scaling [153]. The same system was also employed in the inspection detection and product verification for touch-screen panels [154].

The compressive sensing (CS) reconstruction algorithm SpaRSA [155] associated with a hybrid one-CPU, triple-GPU architecture was introduced to realize a real-time SD-OCT system (CS SD-OCT) [156], which could display B-mode images at 70+ fps. CS SD-OCT took advantage of cuFFT and the parallel reduction. The three GPUs were mainly used to resolve the inconsistent iterations of the A-scans reconstruction, which brought divergent branches to GPU threads. This triple-GPU architecture was also employed in the GPU-accelerated non-uniform FFT-based CS (GPU-NUFFT-CS) SD-OCT system [157], and the FFT-based two-step real-time dispersion-compensated image reconstruction algorithm [158].

Wang et al. proposed a speckle decorrelation algorithm and applied the GPU to accelerate the cross-correlation for robust motion tracking in OCT images [159]. The spatial convolution for mirror image suppression in SD-OCT was powered by the GPU [160].

Swept-source OCT systems. SS-OCT enjoys both a higher imaging speed while providing greater details than SD-OCT. Nevertheless, without parallel computing, the processing speed of SS-OCT can hardly match its acquisition speed [161]. Lee et al. used a hybrid CPU-GPU architecture to implement a real-time speckle variance (SV) SS-OCT system [162]. CUDA streams and cuFFT were employed to implement a real-time SS-OCT Doppler vibrography system for *in vivo* middle ear diagnostics [163]. Viehland et al. demonstrated a real-time 3D volumetric visualization for SS-OCT [164].

The master-slave OCT (MS-OCT) method which eliminated the FFT process was proposed in [165, 166] and parallelized by the GPU [167]. The GPU-accelerated MS-OCT was applied to a scanning laser ophthalmoscopy (SLO) system. MS-OCT

was then improved by the association with a Gabor filter to provide a constant transversal resolution from the investigated samples [168].

2.4.3.2 Other tomography techniques

Diffuse optical tomography (DOT). DOT is a tomographic technique that uses near infrared light. It is mainly used to capture the optical properties of physiological tissues by monitoring spatial-temporal variations in the light absorption and scattering. Solving nonlinear and ill-posed inverse problem is the heaviest computation burden on DOT. Thus, Zhang and Jiang used the GPU to accelerate the FEM-based 3D DOC system [169], in which matrix multiplications were directly carried out by cuBLAS, the construction of the Jacobian matrix was performed in a CUDA kernel, and the LU factorization was optimized by their proposed `gpu_lapack` toolbox. A full domain decomposition (DD) was proposed and parallelized using a hybrid CPU-GPU architecture to accelerate DOT applications in large-sized tissues [170]. Similarly, multiple nodes composed of CPUs and GPUs were used to accelerate the fDOT image reconstruction based on the path-history fluorescence Monte Carlo model [171].

Optical diffraction tomography (ODT). The wavy nature of light is used for 3D reconstruction in ODT [172]. Kim et al. realized a real-time 3D refractive index (RI) distribution reconstruction algorithm based on ODT by computing the sparse angle illumination on the GPU [173]. MATLAB's Parallel Computing Toolbox [174] was used for GPU programming, and FFTs, iFFTs, and matrix multiplications were performed on the GPU. The solution of the complex Helmholtz equations involved in non-linear ODT (NLODT) was accelerated on the GPU for fast determination of the locations of particles in fluid velocimetry [175], in which the sparse matrix vector multiplications (SpMV) were accelerated by cuBLAS.

Photoacoustic tomography (PAT). PAT is a non-invasive modality that is widely used in biomedical imaging to provide physiologically high optical absorption contrast images with high resolution. The ultrasound waves generated from non-ionizing pulsed laser excitation were detected and reconstructed to generate

images [176]. The GPU was used to parallelize the convolution, filtering and reconstruction processes in a real-time back-projection PAT implementation [177]. Peng et al. employed 4 parallel computing strategies to accelerate their FEM-based reconstruction method [178]: (i) An atomic operation was used to construct the sparse element matrices based on a 1D sparse matrix format; (ii) the parallel reduction and the tiling pattern were employed to parallelize the multiply accumulation and matrix multiplication, respectively; (iii) a sparse symmetric linear solve was accelerated on the GPU using the LDLT factorization; (iv) the Jacobian matrix generation and objective function solution were also performed on the GPU. A hybrid CPU-GPU architecture was used for real-time optical-resolution photoacoustic microscopy (OR-PAM) [179].

2.4.4 Hyperspectral imaging

In hyperspectral imaging (HSI), measurement information is collected and processed as 3D data cubes. Each element of a data cube contains a narrow wavelength range of spectral bands. All available information at each point can then be mined from HSI spectrums by a post-processing step without prior knowledge of the sample [180]. However, this post-processing step, including compression, unmixing, image classification, and anomaly detection, requires to process gigabyte hyperspectral cubes, thus is very time-consuming [181]. The CPU and GPU parallel computing strategies applied to these post-processing steps are reviewed as follows.

2.4.4.1 Compression

Compression, dimensionality reduction, and compressive sensing (CS) have been used to compress the large volume of 3D data in HSI. For compression, a parallel JPEG 2000 algorithm was implemented on the GPU [182]. For dimensionality reduction, the principal component analysis (PCA) [183], the parallel optimized maximum noise fraction (G-OMNF) [184], and the band selection algorithm [185] were accelerated on the GPU. GPU implementations of the hyperspectral coded aperture (HYCA) algorithm [186,187] and the spectral compressive acquisition (SpeCA) algorithm [188] were proposed for fast CC.

2.4.4.2 Unmixing

Unmixing is required to extract the exact material types from mixed pixels in HSI images that generated by HSI devices with limited resolutions. The linear unmixing chain composed of estimation of the number of pure spectral endmembers, endmember extraction, and estimation of the fractional abundance of endmembers. To estimate the number of pure spectral endmembers, both the CPU and GPU parallel implementations of the virtual dimensionality (VD) algorithm [189–191] and the hyperspectral signal identification by minimum error (HySime) algorithm [190–195] were proposed. For endmember extraction, the orthogonal subspace projection with Gram-Schmidt orthogonalization (OSP-GS) algorithm [189–191, 196, 197], the famous N-FINDR algorithm [183, 190, 191, 198], the iterative error analysis (IEA) algorithm [191, 199], the pixel purity index (PPI) algorithm [184, 200], the iterated constrained endmember (ICE) algorithm [201], the vertex component analysis (VCA) algorithm [191], the minimum volume simplex analysis (MVSA) algorithm [202], the modified VCA (MVCA) [203], the simplex identification via split augmented Lagrangian (SISAL) algorithm [204], the simplex growing (SGA) algorithm [186], the spatial preprocessing (SPP) [191, 205, 206], and the spatial-spectral preprocessing (SSPP) [207], were implemented on the GPU. For estimating the fractional abundance for each pixel’s endmember, both the unconstrained [183, 189–191, 194, 196] and nonnegativity constrained [190, 197, 200] least-squares (UCLSU and NCLSU, respectively) unmixing algorithm were parallelized.

Multiple GPUs [202] as well as a hybrid CPU-GPU architecture [194] were also attempted to achieve an even higher unmixing speed. Moreover, an open-source unmixing application named HyperMix [191] with many careful optimizations was proposed by Jiménez et al. Sánchez claimed a real-time linear unmixing chain [183]. Recently, Martel et al. demonstrated the GPU implementation of the novel FUN algorithm, pFUN [208], that is becoming the fastest unmixing algorithm so far.

2.4.4.3 Anomaly detection

Anomaly detection is required to detect abnormal spectral features within HIS images without prior knowledge. Based on the widely applied RX algorithm [209] for anomaly detection, the GPU implementations of the global RX [210], the local RX [210–212], and the casual RX [213] algorithms were then proposed.

2.4.4.4 Image classification

Image classification is used to construct land-cover maps from remotely sensed HSI images. Parallel implementations of image classification methods include the automatic target detection and classification (ATDCA) algorithm [214], the extreme learning machine (ELM) algorithm [215], the support vector machine (SVM) based algorithms [216–218], and the Markov random field (MRF) based algorithms [219, 220].

2.4.4.5 On-board unmixing

Besides the above mentioned parallel implementations using ground-based hardware, on-board unmixing algorithms have also been attempted to eliminate the low-bandwidth downlink process. Both the FPGA [221] and the GPUs [222–225] were employed in on-board unmixing applications.

2.4.5 Computer generated holograms

Computer generated holograms (CGH) is used to generate digital holographic fringe patterns. It has been widely applied to 2D, 3D, stereoscopic, and volumetric imaging as well as interferometric measurement of aspheric surfaces due to its capability of change a virtual wavefront to any shape using a computer. Basically, CGH can be generated by the FFT-based methods and the point-based methods, both of which have been well accelerated by the GPU.

For FFT-based methods, FFTs are the most common operations being parallelized (e.g. using cuFFT). The wavefront-recording plane (WRP) was proposed and accelerated on the GPU to improve image quality in large-scale CGHs

[226, 227]. A fast pre-calculated triangular mesh was parallelized to speed up binary CGH calculations [228]. For fast CGH animation containing viewpoint parallel shifts and rotations, a Fourier transform optical system (FTOS) was implemented on the GPU [229]. Good speedups of FFT-based CGHs have also been achieved by directly using the MATLAB's Parallel Computing Toolbox [230] without knowing much parallel computing knowledge.

Point-based methods are more time-consuming than FFT-based ones. Look-up tables have been primarily used for their accelerations [231–233], in which high speed was achieved by a large consumption of memory space. To achieve a real-time frame rate, accelerations by both multiple GPUs [234] and GPU clusters [46] have been attempted. The linear and triangle patch modeling method for CGH were parallelized in [235, 236].

Other GPU-accelerated techniques related to CGH include the hybrid FFT-based and point-based CGH algorithm [237], the GPU-accelerated digital holography used in real-time beam shaping applications [238–240], fast CGH compression [241], and compressive holography [242].

2.4.6 Integral imaging

Integral imaging (II) is a technique to capture and reproduce a light field for auto-stereoscopic 3D imaging. The GPU was employed to implement a II-based incoherent Fourier hologram capture method [243] and a back propagation algorithm associated with an LUT for 3D computational II [244, 245]. In tomography and magnetic resonance imaging (MRI), real-time computer generated II has been realized [246–248].

2.5 Discussion

Table 2.1 summarizes the reviewed CPU and GPU parallel computing works. It has been found that, in the past 5 years since 2012, more than 100 papers have been published on using CPUs and GPUs to accelerate their computationally intensive tasks, which implies that the CPU and GPU parallel computing has become

the mainstream in accelerating optical methods. This is due to the increasing demand of higher measurement accuracy as well as more advanced and complex algorithms. Besides, as the efforts that should be put into parallel programming becomes smaller, parallel computing strategies have been becoming more sophisticated and reusable in applications containing the common parallel patterns introduced in Section 2.2. As optical methods always contain sufficient parallelism as illustrated in the above sections, one can always refer to Sections 2.2 and 2.3 when they come across similar problems which can be accelerated by the existing methods.

Moreover, building up clusters containing nodes of hybrid CPU and GPU or multiple GPUs architectures is a proven simple yet effective way to achieve even higher speed for super computationally complex problems. Because of the excellent scalability and portability of CPU and GPU computation libraries such as MPI, OpenMP, and CUDA, etc., such clusters are guaranteed to obtain a higher computational performance with more nodes of CPUs and GPUs involved.

In the following sections, guided by the knowledge and experience obtained from the above review, we focus on how to design and optimize parallel computing strategies using CPUs and GPUs to realize high-performance, real-time DIC&DVC and phase extraction in FPA.

Table 2.1: Summerization of the reviewed research works in DIC&DVC, FPA, tomography, HSI, CGH, and II.

		Main applications of CPU and GPU	References
DIC&DVC	Subset-based DIC	FFT, ZNCC, ZNSSD, interpolation	[21, 45, 49, 50, 52, 55, 115–126]
	FEM-based DIC	Matrix construction, linear solvers	[59, 128–131]
FPA	Fringe Projection	Phase conversion, 3D registration, rendering	[132–135, 137, 138]
	Other FPA techniques	Phase extraction, WFT	[139–144]
Tomography	OCT	DC removal, resampling, FFT, log scaling, multiple GPU, hybrid CPU and GPU	[145–157, 159–168]
	Other tomography techniques	DOT, ODT, PAT	[169–179]
	HSI	Compression, unmixing, anomaly detection, classification, on-board unmixing	[182–191, 194–197, 199–202, 204–208, 210–225]
	CGH	Look-up table, FFT, rendering,	[46, 226–242]
	II	49 FFT, look-up table, back propagation	[243–248]

Chapter 3

Parallel digital image correlation

In this chapter, based on the path-independent DIC (PiDIC) algorithm [48], which used FFT-CC for initial guess estimation and IC-GN for sub-pixel registration at each POI independently, a parallel DIC (paDIC) algorithm ¹ is proposed and implemented on the GPU using CUDA. The accuracy, precision and computation efficiency of paDIC are verified on both simulated and real images.

3.1 Introduction

Parallel computing based on general purpose graphical units (GP-GPUs) is becoming popular. If all POIs in DIC were processed simultaneously, a dramatic speedup could be expected. The parallelization, in fact, has been successfully realized in integer-pixel DIC methods [49, 59, 115, 116], in which the FFT-CC algorithm has been well accelerated on the GPU. However, as announced in [48], the heaviest computation burden of DIC resides in its iterative sub-pixel registration procedure, yet no parallel computation algorithm has been attempted for its acceleration. Therefore, the proposed paDIC aims at accelerating DIC in both the integer-pixel and sub-pixel procedures, and it has achieved satisfying acceleration results, which will be demonstrated in the next sections of this chapter.

¹This is a joint research work with Lingqi Zhang and Zhenyu Jiang from South China University of Technology, Guangzhou, China.

Figure 3.1: Schematically illustration of the principle of the paDIC algorithm combining the FFT-CC algorithm and the IC-GN algorithm.

3.2 Principle of the paDIC algorithm

The principle of the paDIC is schematically illustrated in Fig. 3.1. At each POI-centric subset, the FFT-CC algorithm is performed to calculate the integer-pixel displacement vector $\mathbf{d} = (u, v)$, which is then fed as an initial guess for the deformation vector $\mathbf{p}_0 = (u, 0, 0, v, 0, 0)$. Thereafter, this initial deformation vector is applied as the initial guess to the subsequent IC-GN algorithm. In each iteration of

the IC-GN algorithm, the calculation of temporary incremental deformation vector $\Delta \mathbf{p}$ and the update of the warp function $\mathbf{W}(\xi; \mathbf{p})$ is repeated until one of the following two convergence conditions is satisfied:

- $\Delta \mathbf{p} = \sqrt{(\Delta u)^2 + (\Delta v)^2} < 0.001$
- The maximum iteration number 20 is reached.

More details of the algorithm can be found in [48]. It is worth reiterating that, however, the initial guess estimated by the FFT-CC algorithm is restricted to small deformation. Thus, large deformation between reference and target images should be avoided technically through increasing the image acquisition rate [51].

Figure 3.2: Schematically illustration of the implementation of the paDIC.

3.3 Implementation of the paDIC algorithm

Based on the CUDA programming model, the implementation of paDIC is shown as Fig. 3.2. At stage 1, the initial integer-pixel deformation parameter $\mathbf{p}_0$ at each POI is estimated by the FFT-CC algorithm using the cuFFT library [110]. The pointwise complex number multiplication operations, as shown in Eq. 1.4, is also well accelerated on the GPU using the pixelwise pattern (Section 2.2.1). In order to occupy most of the GPU resource, $\mathbf{p}_0$'s for all the POIs are calculated and stored at this stage.

Subsequently, the deformation vector $\mathbf{p}$ for a POI is calculated by a parallelized IC-GN algorithm, in which every block is responsible for the computation at one POI. Suppose that a block contains M threads. Initially, all the M threads within one block work cooperatively to load the initial guess $\mathbf{p}_0$ and construct the warped target subset through bicubic interpolation. A precomputed bicubic coefficients look-up table approach [46, 48] is employed here to remove the redundant calculations. Thereafter, the incremental deformation vector $\Delta\mathbf{p}$ and the deformation vector $\mathbf{p}$ which both contains 6 elements are calculated using 6 threads. Finally, only one threads is used to check whether the convergence conditions described above are satisfied or not.

3.4 Experimental study

The paDIC algorithm was programmed using C++ based on CUDA 6.5 and tested on a desktop computer equipped with Intel i5-3570 CPU(3.4GHz), 8.0GB RAM and a NVIDIA GeForce GTX 760 graphics card(1152 CUDA cores, 2 GB 256-bit RAM). In this section, the paDIC is compared in terms of accuracy, precision and computation efficiency with the sequential implementation [48] (seDIC).

3.4.1 Verification of measurement accuracy

In order to quantitatively evaluate the accuracy and precision of the proposed paDIC, an 8-bit grayscale 514×514 reference image containing approximately 4000

Figure 3.3: A simulated speckle pattern used as the reference image in numerical experiments.

random Gaussian speckles with a radius of 2 pixels was generated. Afterwards, twenty target images were generated by translating the reference image along x-axis from 0 to 1 pixels, with a step of 0.05 pixels between every two successive images. A square region of 502×502 was set as the ROI, as shown in Fig. 3.3, in which $157 \times 157 = 24,649$ POIs were computed using a 33×33 -pixel subset.

The accuracy of the paDIC is studied by the mean bias error written as:

$$e_u = \frac{1}{M} \sum_{i=1}^M (u_i - u_d), \quad (3.1)$$

where M denotes the number of POIs; u_i is the calculated displacement at the i th POI, and u_d is the pre-set displacement. The precision of the paDIC is evaluated by standard deviation defined as:

$$\sigma_u = \sqrt{\frac{1}{M-1} \sum_{i=1}^M \left(u_i - \frac{1}{M} \sum_{i=1}^M u_i \right)^2}. \quad (3.2)$$

Figure 3.4 shows the mean-bias error and the standard deviation of the calculated u -components. The mean-bias error falls in a range from -3.3×10^{-3} to 3.2×10^{-3} pixels, and the standard deviation is less than 2.1×10^{-3} pixels, which prove that

the proposed paDIC achieves satisfying accuracy and precision (on the order of 10^{-3}).

Figure 3.4: Mean absolute errors (a) and standard deviations (b) of the calculated u -components. Adapted with permission from [83], Opt Lasers Eng V.69, P.7.

3.4.2 Evaluation of computation efficiency

Computation speeds of paDIC on the simulated images are compared with its sequential counterpart seDIC in Fig. 3.7. Note that the GPU memory allocation and deallocation time is not considered in this experiment since they are only required to be performed once if the parameters and the image size are not changed during the test. As shown in Fig. 3.7, the average computation speed of paDIC is 1.66×10^5 POI/s, which demonstrates a remarkable superiority over seDIC.

The computation efficiency of paDIC on real images is then studied. A series of real experimental images shown in Fig. 3.5 is appreciatively obtained from the website of the Society for Experimental Mechanics (Digital Image correlation Challenge, Sample 12) [249]. The image sequence records the deformation of a plate specimen subjected to a unidirectional tension along x-axis. For the convenience of display, the original images with a size of 400×1040 pixels are rotated by 90 degree clockwise. A 950×270 -pixel ROI is set as the calculation area, within which 24,174 (306×79) POIs with 33×33 -pixel subsets are processed. Also note that

Figure 3.5: A series of real images recording a plate specimen with a hole subjected to unidirectional tension along x-axis (obtained from DIC challenge [249]). The solid-line rectangle outlines the ROI for DIC calculation. Adapted with permission from [83], Opt Lasers Eng V.69, P.7.

the time cost by image reading, memory allocation, memory transfer and memory deallocation is not taken into consideration. The corresponding displacement field along x-axis is shown in Fig. 3.6.

Table 3.1 compares the computation time consumed by the seDIC and paDIC

Figure 3.6: Displacement field along x-axis calculated using paDIC on the real images. Adapted with permission from [83], Opt Lasers Eng V.69, P.7.

Figure 3.7: Comparison of computation speed between paDIC and seDIC. Adapted with permission from [83], Opt Lasers Eng V.69, P.7.

Table 3.1: Statistics of computation time in different stages for the seDIC and the paDIC. Adapted with permission from [83], Opt Lasers Eng V.69, P.7.

		Simulated speckle images			Real speckle images		
		seDIC	paDIC	Speed-up	seDIC	paDIC	Speed-up
Mean computation time [ms]	Overall	8581.09	149.16	57.5×	16367.33	213.87	76.5×
	Pre-computation	46.64	0.46	101.1×	48.21	0.44	109.3×
	Integer-pixel registration	583.93	28.73	20.3×	586.75	28.15	20.8×
	Sub-pixel registration	7981.49	119.31	66.9×	15766.35	184.73	85.4×

for both the simulated and real images. For the simulated speckle images, the average computation speed of the seDIC can be approximately calculated as 2.89×10^3 POI/s, whereas the paDIC achieves 1.66×10^5 POI/s which is about $57\times$ faster. For the real speckle images, the paDIC achieves an average computation speed of 1.13×10^5 POI/s, which is about $76\times$ faster than that of the seDIC (1.48×10^3 POI/s). A video-rate (24 fps) paDIC can thus be realized with up to 4700. It is worth noting that the computation speed of paDIC on the real experimental images is higher than on the simulated images. The reason is that more iterations are required for the convergence of paDIC on the real- experimental images, since they are usually captured with a poorer quality. While the average iteration number is 3.53 for the simulated images, the real experimental images require 7.21 average iterations. As all the time-consuming operations in each iteration are parallelized on the GPU, the more iterations IC-GN requires, the more acceleration can be obtained by paDIC.

3.5 Conclusion

We have demonstrated the implementation of the high accuracy sub-pixel paDIC method powered by the GPU. The computation of paDIC achieves nearly two orders of magnitude speedup than its sequential counterpart on a commodity laptop. Compared with the parallel implementation of the well-known RG-DIC method which was run on a desktop equipped with a similar grade CPU [119], our paDIC

is still obviously a more efficient solution yet without losing accuracy. Further performance improvement can be expected if the proposed paDIC can be run on one or more professional GPUs with more SMs and global memory, which will make the computation efficiency no longer a critical issue in sub-pixel DIC methods and a real-time DIC system can be potentially realized.

Chapter 4

Parallel digital volume correlation

Digital volume correlation (DVC) is a direct 3D extension of the well-established 2D DIC. However, the computation burden of DVC is much heavier than DIC due to the additional dimension. For example, a typical serial implementation of DIC on a typical desktop computer consumes more than 10 seconds to process a 1024×1024 images (approximately 40,000 POIs) with a 5-pixel interval between each two adjacent POIs, while more than 4 days are required by DVC to calculate one $1024 \times 1024 \times 1024$ volume with the same configuration (approximately 8,000,000 POIs).

In this chapter, as an extension of the paDIC algorithms, a parallel DVC (paDVC) algorithm, which greatly improves the computation efficiency of DVC without compromising with accuracy and precision, is proposed. However, due to the heavier computation burden introduced by the extra dimension, more advanced parallel computing strategies and optimizations for paDVC are employed and explained. Finally, the experimental results of the accuracy and computation efficiency of paDVC are verified on simulated volume images.

As illustrated in Section 2.4.1, efforts in accelerating DVC have also been made recently. In this chapter, the paDVC algorithm which is the extension of the paDIC algorithm is proposed and assessed using computer generated 3D speckle pattern volumes. Although paDVC is similar to paDIC, more complexities are introduced by the additional dimension. Therefore, for the sake of clear illustration and making this chapter self-contained, the principle and implementation of paDVC are

reiterated. Moreover, details regarding the parallel strategies and optimizations applied to all three stages of paDVC, namely precomputation, 3D FFT-CC, and 3D IC-GN as well as the batch processing scheme for handling large-scale 3D volumes are provided.

4.1 Introduction

Since first proposed by Bay and Smith [113, 250], DVC has been adopted as a simple yet effective technique for volumetric deformation measurement within solids. As shown in Fig. 4.1, DVC tracks the motion of a POI $\mathbf{P}(x_0, y_0, z_0)$ in the reference volume image (Fig. 4.1. (a)) to a point $\mathbf{P}'(x'_0, y'_0, z'_0)$ in the target volume image (Fig. 4.1. (b)). In order to avoid mis-registration, a cubic subvolume centered at $\mathbf{P}(x_0, y_0, z_0)$ is selected as the basic matching unit. As a direct extension to DIC [22], DVC shares its simplicity in principles and effectiveness in applications, and thus has been successfully applied in the analysis of various materials including bones [113, 251], soft materials [33, 34], wood [252] and sand [253]. However, the computation burden of DVC is much heavier than DIC. Increasing the computation efficiency of DVC have thus become a challenging research topic in the past decade.

Similar to DIC, DVC is performed at two levels: an initial integer-voxel level displacement estimation followed by a sub-voxel level refinement. In terms of integer-voxel displacement estimation, Franck et al. suggested a 3D fast Fourier transform based cross-correlation (3D FFT-CC) algorithm [33], which benefits from the fact that the cross-correlation operation in space domain is equivalent to point-wise multiplication in frequency domain. Alternatively, a fast sum-table approach [34] can be applied to calculate cross-correlation efficiently in space domain. To pursue a higher sub-voxel accuracy, various sub-voxel registration algorithms have been proposed, including the correlation coefficients curve-fitting algorithm [33], the iterative Gauss-Newton algorithm [113] as well as its variant the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [35, 125], the gradient-based algorithm [34], and the iterative least-squares algorithm [254].

Figure 4.1: Schematic of the principle of DVC: the displacement vector $\mathbf{d} = (u, v, w)^T$ from a POI $\mathbf{P}(x_0, y_0, z_0)$ in the reference volume to $\mathbf{P}'(x'_0, y'_0, z'_0)$ in the target volume is obtained by tracking the corresponding cubic subvolumes. $\mathbf{Q}(x_i, y_i, z_i)$ and $\mathbf{Q}'(x'_i, y'_i, z'_i)$ indicate arbitrary points within the reference subvolume and the target subvolume, respectively. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

To improve the computation efficiency of DVC algorithms, Pan et al. proposed three smart ideas in implementing a DVC algorithm [255]: (i) a 3D ICGN algorithm was introduced to substitute the for sub-voxel registration, which eliminates the redundant calculations within the iterative procedure (Section 1.3.2.2); (ii) the same reliability-guided displacement tracking strategy, which has been successfully used in DIC [47], was employed to provide accurate and robust initial guesses for the 3D ICGN algorithm; (iii) a global LUT of tri-cubic interpolation coefficients [53] was pre-established to eliminate the redundant calculation of sub-voxel interpolation coefficients for the reconstruction of the warped target subvolumes during the ICGN iterations. The computation speed of DVC employing these methods surges up from about 0.9 POIs per second (POIs/s) to 41.1 POI/s using a $19 \times 19 \times 19$ -voxel sub-volume [255]. Further speedup can be substantially achieved by processing POIs in parallel, however, the employed path-dependent reliability-guided initial

guess transfer strategy leads to a sequential algorithm. To overcome this difficulty, our preliminary research proposed the PiDIC algorithm [48] (Section 3.3), which estimated the initial guess for ICGN at each POI independently using FFT-CC. Elimination of path-dependence rendered the application of parallel computing to realize the paDIC algorithm [51], which has achieved a nearly two orders of magnitude of speedup without compromising high accuracy and precision (Section 3.5).

4.2 Principle of the paDVC algorithm

The three-stage paDVC algorithm is schematically illustrated in Fig. 4.2. The integer-voxel displacement estimation is first calculated by the 3D FFT-CC algorithm at each POI-centric subvolume. The obtained integer-voxel deformation parameter vector $\mathbf{p}_0 = (u, 0, 0, 0, v, 0, 0, 0, w, 0, 0, 0)^T$ is fed as the initial guess into the subsequent 3D IC-GN iterations for the sub-voxel refinement. Prior to these two stages, the inverse of the Hessian matrix $\mathbf{H}_{12 \times 12}^{-1}$ for each reference subvolume as well as a global tri-cubic interpolation coefficients LUT are precomputed, which enormously decrease the number of redundant calculations during IC-GN iterations.

4.2.1 Integer-voxel estimation using 3D FFT-CC

Defining a sub-volume centered at a POI containing $N = (2M + 1) \times (2M + 1) \times (2M + 1)$ voxels, the displacement vector $\mathbf{d} = (u, v, w)^T$ (see Fig. 4.1) can be obtained by searching the peak position of the ZNCC coefficients:

$$C_{ZNCC}(u, v, w) = \frac{\sum_i (R_i - R_m)(T_i - T_m)}{\sqrt{\sum_i (R_i - R_m)^2 \sum_i (T_i - T_m)^2}}. \quad (4.1)$$

Figure 4.2: Flow chart of the proposed paDVC, where R and T refer to the reference sub-volume and target sub-volume, respectively, ξ is the local coordinates of points within the sub-volume. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Similar to Eq. 1.3, based on the Fourier theorem, the calculation of Eq. 4.1 can be performed as a simpler point-wise multiplication in frequency domain, i.e.

$$C_{ZNCC}(u, v, w) = FFT^{-1} \left\{ FFT^* \left[\frac{\sum_i (R_i - R_m)}{\sqrt{\sum_i (R_i - R_m)^2}} \right] \cdot FFT \left[\frac{\sum_i (T_i - T_m)}{\sqrt{\sum_i (T_i - T_m)^2}} \right] \right\}. \quad (4.2)$$

The integer-voxel level displacement vector $\mathbf{d} = (u, v, w)$ is determined from the indices of the peak $C_{ZNCC}(u, v, w)$ and then filled into the initial deformation parameter vector $\mathbf{p}_0 = (u, 0, 0, 0, v, 0, 0, 0, w, 0, 0, 0)^T$.

4.2.2 Sub-voxel registration using 3D IC-GN

Assume that an arbitrary point $\mathbf{Q}(x_i, y_i, z_i)$ in the reference subvolume R deforms to a point $\mathbf{Q}'(x'_i, y'_i, z'_i)$ in the target sub-volume T (see Fig. 4.1), the deformation map from $\mathbf{Q}$ to $\mathbf{Q}'$ can be described by the first-order shape function as

$$\begin{aligned} x'_i &= x_i + u + u_x \Delta x + u_y \Delta y + u_z \Delta z \\ y'_i &= y_i + v + v_x \Delta x + v_y \Delta y + v_z \Delta z. \\ z'_i &= z_i + w + w_x \Delta x + w_y \Delta y + w_z \Delta z \end{aligned} \quad (4.3)$$

Equation 4.3 can be rewritten in a matrix form as

$$\mathbf{Q}' = \mathbf{P} + \mathbf{W}(\xi, \mathbf{p}), \quad (4.4)$$

where $\mathbf{W}(\xi, \mathbf{p})$ represents the 3D warp function,

$$\mathbf{W}(\xi, \mathbf{p}) = \begin{bmatrix} 1 + u_x & u_y & u_z & u \\ v_x & 1 + v_y & v_z & v \\ w_x & w_y & 1 + w_z & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ 1 \end{bmatrix}, \quad (4.5)$$

where $\xi = (\Delta x, \Delta y, \Delta z, 1)^T$ is the local coordinates of point $\mathbf{Q}$ within the reference sub-volume in homogeneous form, $u_x, u_y, u_z, v_x, v_y, v_z, w_x, w_y$, and w_z are the gradients of u, v , and w in x, y , and z directions, respectively; $\mathbf{p} = (u, u_x, u_y, u_z, v, v_x, v_y, v_z, w, w_x, w_y, w_z)^T$ represents the vector of deformation parameters. Besides, the incremental warp function $\mathbf{W}(\xi, \Delta \mathbf{p})$ used to adjust the shape of the reference sub-volume is written as [255]

$$\mathbf{W}(\xi, \Delta \mathbf{p}) = \begin{bmatrix} 1 + \Delta u_x & \Delta u_y & \Delta u_z & \Delta u \\ \Delta v_x & 1 + \Delta v_y & \Delta v_z & \Delta v \\ \Delta w_x & \Delta w_y & 1 + \Delta w_z & \Delta w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ 1 \end{bmatrix}, \quad (4.6)$$

where $\Delta \mathbf{p} = (\Delta u, \Delta u_x, \Delta u_y, \Delta u_z, \Delta v, \Delta v_x, \Delta v_y, \Delta v_z, \Delta w, \Delta w_x, \Delta w_y, \Delta w_z)^T$ is the incremental vector of deformation parameters. The $\Delta \mathbf{p}$ can then be determined using the 3D ICGN algorithm which minimizes the ZNSSD criterion as

$$\Delta \mathbf{p} = \mathbf{H}_{12 \times 12}^{-1} \sum_{\xi} \left\{ \left[\nabla R \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[\frac{\bar{R}}{\bar{T}} (T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})] - T_m) - (R(\mathbf{P} + \xi) - R_m) \right] \right\}, \quad (4.7)$$

where ∇R is the gradient within the reference sub-volume,

$$\nabla R = \left(\frac{\partial R(x, y, z)}{\partial x}, \frac{\partial R(x, y, z)}{\partial y}, \frac{\partial R(x, y, z)}{\partial z} \right); \quad (4.8)$$

$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the Jacobian matrix of the warp function,

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} 1 & \Delta x & \Delta y & \Delta z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta x & \Delta y & \Delta z & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta x & \Delta y & \Delta z \end{bmatrix}; \quad (4.9)$$

and $\mathbf{H}_{12 \times 12}^{-1}$ denotes the inverse of the 12×12 Hessian matrix $\mathbf{H}$,

$$\mathbf{H} = \sum_{\xi} \left\{ \left[\left(\nabla R \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right)_{12 \times 1}^T \left(\nabla R \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right)_{1 \times 12} \right] \right\}. \quad (4.10)$$

The obtained $\Delta \mathbf{p}$ is then substituted back into Eq. 4.6 to obtain the incremental warp function $\mathbf{W}(\xi, \Delta \mathbf{p})$, which is then used to update the warp function $\mathbf{W}(\xi, \mathbf{p})$ as

$$\mathbf{W}(\xi, \Delta \mathbf{p}) \leftarrow \mathbf{W} [\mathbf{W}^{-1}(\xi, \Delta \mathbf{p}), \mathbf{p}] = \mathbf{W}(\xi, \Delta \mathbf{p}) \mathbf{W}^{-1}(\xi, \Delta \mathbf{p}), \quad (4.11)$$

where

$$\mathbf{W}^{-1}(\xi, \Delta \mathbf{p}) = \begin{bmatrix} 1 + \Delta u_x & \Delta u_y & \Delta u_z & \Delta u \\ \Delta v_x & 1 + \Delta v_y & \Delta v_z & \Delta v \\ \Delta w_x & \Delta w_y & 1 + \Delta w_z & \Delta w \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ 1 \end{bmatrix} \quad (4.12)$$

is the inverse of the incremental warping function. The procedures shown in Eqs. 4.7 and 4.11 are repeated until one of the following two convergence criteria is satisfied

- $\sqrt{(\Delta u)^2 + (\Delta v)^2 + (\Delta w)^2} < 0.001$
- The maximum iteration number reaches 20.

It is clear that ∇R , $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$, and $\mathbf{H}$ as well as its inverse $\mathbf{H}^{-1}$ only depend on the reference subvolume. Since the reference subvolume is not modified by ICGN, these quantities thus can be precomputed.

The reconstructed $T[\mathbf{P} + \mathbf{W}(\xi; \mathbf{p})]$ may reside at a sub-voxel location (point C shown in Fig. 4.3). Therefore, an interpolation is also required to estimate the sub-voxel intensities. As cubic spline interpolations provide a lower systematic error, it is always preferred to polynomial interpolations in DIC [256]. However, the tri-cubic interpolation is still favored in paDVC based on the following two considerations: (i) the computation burden of DVC is several orders of magnitude heavier than that of DIC. Therefore, the computationally more efficient tri-cubic interpolation which can even be accelerated by a precomputed interpolation coefficients LUT can reduce the interpolation time by orders of magnitude with an acceptable and always imperceptible accuracy loss; (ii) in our preliminary research, the proposed PiDIC algorithm [48] using the bi-cubic interpolation achieved an excellent accuracy and precision comparable with the NR algorithm using the bicubic spline interpolation reported in [42]. Furthermore, it has been proved that, in real experiments, the difference between the bias errors of using the NA algorithm with the bicubic interpolation and the quantic spline interpolation can be reduced to a negligible level by employing a Gaussian pre-filtering process [257]. In other words, the tri-cubic interpolation with a global LUT is a good trade-off between computation efficiency and estimation accuracy for paDVC.

4.3 Implementation of the paDVC algorithm

4.3.1 Precomputation

The gradients ∇R and ∇T used in building the inverse of the Hessian matrix in the reference sub-volume and the tri-cubic interpolation coefficients look-up table for

constructing the target sub-volume are calculated in terms of the central difference scheme,

$$\begin{aligned}
 \frac{\partial R(x, y, z)}{\partial x} &= \frac{R(x+1, y, z) - R(x-1, y, z)}{2} \\
 \frac{\partial R(x, y, z)}{\partial y} &= \frac{R(x, y+1, z) - R(x, y-1, z)}{2} \\
 \frac{\partial R(x, y, z)}{\partial z} &= \frac{R(x, y, z+1) - R(x, y, z-1)}{2} \\
 \frac{\partial T(x, y, z)}{\partial x} &= \frac{T(x+1, y, z) - T(x-1, y, z)}{2} \\
 \frac{\partial T(x, y, z)}{\partial y} &= \frac{T(x, y+1, z) - T(x, y-1, z)}{2} \\
 \frac{\partial T(x, y, z)}{\partial z} &= \frac{T(x, y, z+1) - T(x, y, z-1)}{2}.
 \end{aligned} \tag{4.13}$$

It indicates that the computation of the gradient at a voxel involves the intensity values of $3 \times 3 \times 3$ voxels. As illustrated in Section 2.2.2, these neighboring operations can be well parallelized by the tiling pattern. Thus, instead of calculating the gradient at each individual voxel, $6 \times 6 \times 6 = 216$ voxels are grouped to be calculated using CUDA blocks containing $8 \times 8 \times 8$ threads. As shown in Fig. 4.4, the intensity values of $8 \times 8 \times 8$ voxels are loaded into the shared memory allocated to each block. Afterwards, the calculation of gradients at 216 voxels are performed by 216 threads within a block in parallel. The other 296 ($512-216$) threads are deactivated after storing the intensity values at voxels along the outer boundary. This tiled implementation trades memory space for faster running time. All the required data are loaded only once, thus minimizing the GPU global memory access, which is quite time-consuming (Section 2.1.2). Moreover, the gradient precomputation kernel achieves a 100% occupancy on GPU SMs, indicating that the GPU can potentially achieve its peak performance.

The tri-cubic interpolated intensity values used in Eq. 4.7 are calculated as

$$T(x, y, z) = \sum_{i=0}^3 \sum_{j=0}^3 \sum_{k=0}^3 \alpha_{ijk} x^i y^j z^k, \tag{4.14}$$

where the 64 coefficients α_{ijk} are obtained from the intensity values of its eight surrounding integer voxels as well as their gradients (see Fig. 4.3). As shown in

Figure 4.3: Diagram of the tri-cubic interpolation at a sub-voxel position surrounded by 8 integer voxels.

Figure 4.4: Schematically illustration of the tiled computation for volume image gradients. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

[258, 259], α_{ijk} can be solved by

$$\alpha_{ijk} = [\mathbf{A}_{64 \times 64}^{-1} \mathbf{x}]_{ijk}, \quad (4.15)$$

where $\mathbf{A}^{-1}$ is a 64×64 constant matrix that can be directly used; $\mathbf{x}$ is a vector containing the 64 elements of the intensity values and the gradients at the 8

surrounding integer voxels. Therefore, the LUT can also be computed using the similar tiling pattern, in which each $8 \times 8 \times 8$ thread block computes the α_{ijk} 's for $7 \times 7 \times 7$ 8-voxel units. The constructed LUT is stored in the GPU's global memory for the 3D ICGN algorithm.

4.3.2 Parallel 3D FFT-CC

The 3D FFT-CC algorithm in each subvolume is implemented in two steps: (i) calculating $C_{ZNCC}(u, v, w)$ in frequency domain using FFT and FFT^{-1} ; and (ii) searching the positions of peak $C_{ZNCC}(u, v, w)$ values.

The calculation of $C_{ZNCC}(u, v, w)$ coefficients is accelerated by the cuFFT library [110]. In order to maximally utilize the GPU computation resource, subvolumes are grouped in batches using the advanced data layout offered by cuFFT. Although not all of the subvolumes are guaranteed to be executed simultaneously due to hardware limitation, the optimizations taken by cuFFT automatically guarantees a maximum number of subvolumes being processed in parallel.

While the serial implementation of peak position locating is an exhaustive search algorithm, it is accelerated on the GPU by a classical parallel sum reduction algorithm [260], which is schematically illustrated in Fig. 4.5. First, for convenience, a subvolume matrix containing $p \times p \times p$ voxels is transformed into a 1D array. The 3D index (u, v, w) is coded in a 1D index i as

$$i = (w \times p + v) \times p + u, \quad (4.16)$$

which can then be decoded according to

$$u = i \bmod p, \quad v = (i/p) \bmod p, \quad w = (i/p)/p. \quad (4.17)$$

where *mode* represents the modular operator.

Second, CUDA blocks containing m threads ($m = 256$ in the proposed implementation) are allocated to handle all the subvolumes. Before the calculation starts, the $C_{ZNCC}(u, v, w)$ values as well as their 1D indices are loaded into each block's shared memory. It is worth noting that, however, $p \times p \times p$ is usually

Figure 4.5: Flow chart of the parallel reduction algorithm within an individual block. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

greater than the maximum allowed number of threads (i.e. 1024) in a CUDA block. Thus, in practice, the $p \times p \times p$ $C_{ZNCC}(u, v, w)$ values and their indices are divided into m groups, each of which is processed by one thread to obtain the maximum $C_{ZNCC}(u, v, w)$ value of that group by the exhaustive search as used in the serial case.

Third, the parallel sum reduction is executed to locate the final peak $C_{ZNCC}(u, v, w)$ value among the m $C_{ZNCC}(u, v, w)$ values stored in the threads. As shown in Fig. 4.5, in each reduction step, the $C_{ZNCC}(u, v, w)$ values of the first half of the threads within a block is compared with those in the second half. If any of the $C_{ZNCC}(u, v, w)$ values from the first half are smaller than those of its counterpart in the second half, the corresponding two threads exchange their $C_{ZNCC}(u, v, w)$ values as well as their 1D indices. In the next reduction step, the

comparison and data exchange only occur in the active half of threads containing larger $C_{ZNCC}(u, v, w)$ values. This procedure is repeated until only one thread within a block remains active. Finally, the peak $C_{ZNCC}(u, v, w)$ values and corresponding 1D indices for each sub-volume are obtained by the first threads from its corresponding block. The 3D indices can then be restored using Eq. 4.17.

Compared with the sequential exhaustive searching algorithm, which has a theoretical computation complexity of $O(p^3)$ for a $p \times p \times p$ -voxel sub-volume, the parallel reduction algorithm achieves a level of $O[(p^3/m) + \log p^3]$, where m is the number of threads within one block. In the case of $p^3 = m$, the complexity is $O(\log p^3)$.

4.3.3 Parallel 3D IC-GN

Figure 4.6 shows the implementation of the parallel 3D IC-GN algorithm. The computation at each POI-centric subvolume is also allocated by a CUDA block with m threads ($m = 256$ in the proposed implementation). First, the integer-voxel deformation parameter vector $\mathbf{p}$ previously estimated by FFT-CC is loaded in the block's shared memory by the first 12 threads. Second, all the m threads within the block cooperatively construct the warped target subvolume $\mathbf{W}(\xi, \mathbf{p})$ using the precomputed global LUT of tri-cubic interpolation coefficients. Third, the same first 12 threads are used to update the incremental deformation parameter $\Delta\mathbf{p}$. Last, only the first thread remains active to verify if the two convergence criteria have been satisfied and output the final deformation parameter $\mathbf{p}$ after the IC-GN iteration is completed.

4.3.4 Batch processing scheme

Note that the calculation of a large number of POIs with large sub-volume size occupies vast memory space so that it is impossible to process all these POIs in a single process on GPU. For example, for a medium size volume containing $512 \times 512 \times 512$ voxels, dozens of gigabytes of data would be generated during the paDVC method, which is even much larger than the global memory size of a

Figure 4.6: Diagram of the parallel 3D IC-GN algorithm. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

high-end GPU. Thus, to handle these bulky 3D volumetric data, as shown in Fig. 4.7, a batch processing scheme is proposed by first dividing the POIs distributed in the entire volume into a certain number of batches, each of which is surrounded by a small data cube and then processed in turn by the GPU. The size of the batch depends on the spatial sampling rate (i.e. the number of POIs), the size of the subvolumes, and the size of the available global memory of the employed GPU. To

minimize the high-latency global memory accesses, a basic rule in determining the batch size is a maximum occupation of the global memory while all the required data for the computation of one batch of POIs can be stored into the global memory in a single loading. However, the exact batch size is not easy to be obtained analytically. Therefore, in the proposed implementation, a trial-and-error approach is employed since the batch size should be in the form of $S = i^3$, where i is a positive integer. Every time before the execution of paDVC, a set of batch sizes starting from 3^3 is tested with its required memory space needed to be allocated on the GPU. The largest S that can pass the test is set as the final batch size.

Figure 4.7: Schematic of the batch processing scheme used in paDVC.

It is worth noting that, however, the employment of the proposed batch processing scheme has a negative effect on the computation efficiency of paDVC due to the increased number of memory transfers between the primary memory on the main board and the GPU's global memory and the slightly redundant calculations to deal with the boundary of data cube. However, this scheme provides paDVC with a flexible usability and a good scalability to large scale DVC experiments. An obvious potential speedup can be simply achieved by increasing the number of POIs within each batch if a higher-end GPU is used or distribute the batches to different GPUs if a multi-GPU computation node is available.

4.4 Experimental study

4.4.1 Experimental specification

The proposed paDVC is written using C++ and CUDA 6.5, and tested on a Dell workstation equipped with an Intel Xeon CPU E5-1650 with 3.20 GHz main frequency, 16 GB RAM and a NVIDIA GeForce GTX 680 GPU (8 SMs with 1536 CUDA cores and 2GB 256-bit RAM).

As shown in Fig. 4.8, firstly, an 8-bit grayscale reference volume image $R(x, y, z)$ is generated to quantitatively evaluate the accuracy, precision, and computation efficiency of paDVC, using the following equation [255]:

$$R(x, y, z) = \text{round} \left\{ \sum_{i=1}^s I_i \exp \left[-\frac{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2}{r^2} \right] \right\}, \quad (4.18)$$

where $\text{round}(x)$ obtains the nearest integer to x ; s is the total number of random speckles, I_i and (x_i, y_i, z_i) represent the peak intensity value and the center position of the i th speckle, respectively, and r is the radius of the speckles. Subsequently, ten target volume images $T_i(x, y, z)$ are generated by translating the $R(x, y, z)$ along the z -axis with pre-set sub-voxel displacements ranging from 0 to 1 voxel based on the Fourier theorem [256], with a step size 0.1 voxels between each two consecutive images. A random Gaussian noise with a zero mean and a variance of 4 is then added to each of the volume images to simulate the noise. In the experiment, 11 volumes containing $512 \times 512 \times 512$ voxels are constructed, each of which approximately contains 1500,000 speckles with a radius of 4. Totally 729,000 POIs are calculated within one $512 \times 512 \times 512$ volume. The number of POIs processed in one batch is set as 3375, 1000 and 216 for sub-volume sizes of $17 \times 17 \times 17$, $25 \times 25 \times 25$ and $33 \times 33 \times 33$, respectively. An extra safe distance of 5 voxels is set between each neighboring POIs.

4.4.2 Verification on accuracy and precision

The accuracy and precision of the paDVC is examined by the same mean-bias error and standard deviation criteria (see Section 3.4.1) of the w -component.

Figure 4.8: Computer simulated reference volume image $R(x, y, z)$ and the 10 translated target volume images from $T_1(x, y, z)$ to $T_{10}(x, y, z)$. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Figure 4.9 shows the mean bias errors and standard deviations of sub-voxel displacement along the z -axis calculated by paDVC using subvolume sizes of $17 \times 17 \times 17$, $25 \times 25 \times 25$ and $33 \times 33 \times 33$, respectively. It can be observed that the mean bias error of the calculated w -component falls in a narrow range from -4.3×10^{-4} to 2.9×10^{-4} voxels. The variation of subvolume sizes exerts trivial influence on the accuracy of paDVC. The standard deviations of the measured w -component are less than 1.5×10^{-4} , and a larger subvolume size results in a smaller standard deviation for the measurement of sub-voxel displacements. Figure 4.10 gives the mean bias errors and standard deviations calculated on the same series of volume images without noises. It can be found that the introduction of white Gaussian noises reduces the accuracy and precision insignificantly.

4.4.3 Verification on computation efficiency

To assess the computation efficiency of the paDVC, the same DVC algorithm with sequential implementation (seDVC) and multi-threaded implementation (muDVC) by openMP [95] are used as benchmarks.

In the seDVC, all the POIs are processed iteratively. The FFTW library [107] is employed to perform FFT, and the peak C_{ZNCC} values as well as their indices

Figure 4.9: Mean-bias errors (a) and standard deviation (b) of the measured w -components on noise-contaminated volumes calculated by paDVC using different subvolume sizes. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Figure 4.10: Mean-bias errors (a) and standard deviation (b) of the measured w -components on noise-free volumes calculated by paDVC using different subvolume sizes. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

are obtained by a simple exhaustive algorithm. The muDVC implementation is almost identical to seDVC except that a coarse-grained parallelization is used in

the pre-computation step as well as the FFT-CC and the IC-GN algorithms over all the POIs. The number of threads used by openMP is set to be equal to twice the number of physical CPU cores since the hyper-threading technology allows each CPU core to operate at most two simultaneous threads. To make the comparison fair, the batch processing scheme is also adopted in seDVC and muDVC, and the Streaming Single Instruction Multiple Data Extension 3 (SSE3) is enabled for optimizing the CPU code.

Table 4.1 gives the comparisons of the computation time and computation speed required by paDVC, muDVC, and seDVC with respect to different subvolume sizes. It can be observed that paDVC achieves significant speedups, which are approximately $18.3 \times$ $23.3 \times$ faster than seDVC and $3.0 \times$ $3.7 \times$ faster than muDVC. In comparison with another IC-GN algorithm-based sequential implementation, which was tested on a desktop computer with similar a grade CPU [255], paDVC also demonstrates an obvious speed advantage. Increasing the subvolume size effectively reduces the iteration numbers required by the 3D IC-GN algorithm, however, a larger subvolume does not lead to a faster computation speed due to the surge of voxels involved.

Table 4.1: Comparison of the average consumed time and computation speed among paDVC, muDVC, and seDVC. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Subvolume [voxel]	Average consumed time [s]			Average number of iterations			Calculation speed [POI/s]		
	paDVC	muDVC	seDVC	paDVC	muDVC	seDVC	paDVC	muDVC	seDVC
$17 \times 17 \times 17$	416.5	1261.5	7617.5	4.12	4.12	4.12	1750.4	577.9	95.7
$25 \times 25 \times 25$	861.9	2887.9	17862.6	2.91	2.91	2.91	845.8	252.4	40.8
$33 \times 33 \times 33$	1641.4	6061.0	38186.6	2.64	2.64	2.64	444.1	120.3	19.1

Table 4.2 compares the average computation time per batch in the precomputation stage consumed by the three DVC programs. The paDVC implementation is approximately 9.8 and 54.9 faster than muDVC and seDVC, respectively.

Table 4.3 further explores the average computation time consumed by processing one POI in stage 2 (3D FFT-CC) and stage 3 (3D IC-GN). The GPU-powered

Table 4.2: Comparison of the average computation time consumed per batch in the precomputation step among the paDVC, the muDVC and the seDVC, where batch size is the number of POIs processed in one batch; cube size means the size of the small cube enclosing one batch of POIs. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Subvolume [voxels]	Cube Size [voxels]	Batch size	Average computation time consumed by pre-computation [ms/batch]		
			paDVC	muDVC	seDVC
$17 \times 17 \times 17$	$92 \times 92 \times 92$	3375	52.33	498.84	2776.43
$25 \times 25 \times 25$	$75 \times 75 \times 75$	1000	26.75	261.28	1482.43
$33 \times 33 \times 33$	$63 \times 63 \times 63$	216	15.31	152.95	862.77

Table 4.3: Comparison of computation efficiency for 3D IC-GN algorithm between paDVC, muDVC, and seDVC. Adapted with permission from [99], Exp Mch (2016) V56, I2, 297.

Subvolume [voxels]	Average computation time consumed by the 3D FFT-CC algorithm per POI [ms/POI]			Average computation time consumed by the 3D IC-GN algorithm per POI [ms/POI]		
	paDVC	muDVC	seDVC	paDVC	muDVC	seDVC
$17 \times 17 \times 17$	0.005	0.030	0.088	0.501	1.55	9.02
$25 \times 25 \times 25$	0.025	0.114	0.550	1.104	3.40	21.18
$33 \times 33 \times 33$	0.042	0.235	0.740	2.146	7.31	47.75

3D FFT-CC gains a speedup ranging from $4.6\times$ to $22.0\times$ over two CPU-based implementations. The GPU accelerates the 3D IC-GN algorithm for about $3.2\times$ to $19.8\times$ over muDVC and seDVC. The acceleration achieved by paDVC is not only attributed to the coarse-grained parallelism that many POIs are processed simultaneously, but also the fine-grained parallelisms implemented for the computation within each block. It is worth mentioning that, the precomputation stage receives the most remarkable acceleration. This is because it can be performed with high

parallelism, since each CUDA block can calculate the gradients for 216 voxels and the tri-cubic interpolation coefficients for 147 8-voxel units simultaneously, while at most 12 of such operations can be performed in muDVC. In stages 2 and 3, however, the speedup achieved by paDVC dramatically degrades due to the limited number of SPs in the GPU and the complex per-thread operations, resulting in only several POIs can be processed in parallel.

4.5 Conclusion

In this work, a parallel digital volume correlation (paDVC) method is proposed and implemented on the GPU. With the introduction of the 3D FFT-CC algorithm to estimate the integer-voxel initial guess and the 3D ICGN algorithm for sub-voxel registration, the path-dependence of the conventional initial guess transferring scheme used in iterative DVC algorithms is eliminated and thus triggers the applications of parallel computing to accelerate DVC methods. Various parallel computing strategies and optimizations are employed in the proposed paDVC algorithm, achieving a more than $20\times$ speedup than the CPU serial implementation of the same algorithm without compromising with accuracy and precision. Compared with the CPU multi-core implementation, paDVC also demonstrates a superiority with a 3+ times improvement in speed. Therefore, the GPU is a preferred option to accelerate DVC than the CPU. Additionally, paDVC also provides the end-users a possibility of performing large-scale DVC experiments on any of their computers equipped with a CUDA-capable GPU card.

Chapter 5

Development of a hybrid CPU-GPU real-time digital image correlation system

The proposed paDIC and paDVC algorithms have achieved high speedup compared to conventional DIC and DVC methods, and they enjoy the potential to be integrated to realize real-time optical measurement systems. In fact, commercialized DIC systems (e.g. the Ncorr [55] and the VIC-2D [261]) have been developed and successfully applied to multiple applications in areas of experimental mechanics, biomechanics, and material science, etc. However, details of how these systems are designed and implemented have not been studied and exposed yet. Therefore, in this chapter, some thoughts on designing an accurate, real-time, and flexible DIC system based on the proposed paDIC and paDVC algorithms is proposed.

5.1 Introduction

Initially, DIC was invented to extract deformation between a set of two statically captured images representing the reference and the deformed (or target) configurations of a measurement process. However, it is quickly discovered that the behaviors of an object or a material cannot be well characterized by only analyzing an initial state and a final state. Thus, DIC has also been applied to study dynamic phenomena [262], such as the fracture [263], the strain under dynamic

loading[24, 118], and the velocity field [264], by incorporating more images at intermediate states. Nonetheless, due to the high computation burden of conventional DIC methods [40], these dynamic phenomena though can be captured at a rapid speed, have to be processed off-line. To achieve an on-line processing rate, a high-speed DIC method is thus required.

Since DIC assumes small deformation in principle, static DIC analysis also brings the problem of decorrelation when the deformation between the reference and deformed configurations is large. To compensate large deformation which is very common in measurements of soft materials, two categories of methods have been attempted, namely the direct methods [265–268] and the incremental methods [55, 269, 270].

The direct methods only need two images, i.e. one reference image and one target image, thus they can be performed statically. In order to converge to a specified measurement accuracy, the direct methods require highly reliable initial guesses. The feature detection and matching method with scale-invariant feature transform (SIFT) features originated from computer-vision was first applied to the initialization stage [265–267]. As long as more than 3 corresponding points are matched between the reference and the target subsets, the initial guesses can be effectively and reliably calculated. Recently, Zhong and Quan proposed a method to resolve large rotation by transforming the rotation in Cartesian coordinates to the translation in polar coordinate, after which the initial guesses were calculated based on the gradient orientation error between the gradient orientation at the seed points and those at the search points [268]. They claimed that this method could be applied to arbitrary rotation angles. It is worth noting that, however, the direct methods are only applicable to large scaling or rigid rotational deformation, but they are noneffective in resolving significant uni-axial tension or compression, in which case SIFT may fail to even generate a set of corresponding feature points, and the deformation cannot be effectively characterized in polar coordinate.

The incremental methods mitigate large deformation by inserting intermediate images between the reference and target images to make the deformation between

each two consecutive images small enough that the existing DIC algorithms can be applied. The final deformation is then calculated by accumulating the results of all these images. To avoid decorrelation, the reference image is updated if the deformation is getting too large [269] or the correlation coefficients are less than a certain threshold [55, 270]. However, the growth of the number of involved images resulted in a much heavier computation burden to DIC. In fact, as the development of more computationally efficient DIC methods (see Section 2.4.1), the incremental methods are more generally applicable than the direct methods, and the open-source DIC system Ncorr [55] has successfully integrated an incremental method to mitigate large deformation.

Based on the above descriptions, there is a heavy demand of high computation speed in DIC applications. Indeed, several commercialized DIC systems have employed the CPU parallel computing for acceleration [55, 261] and the GPU is also being attempted. However, details of how these systems were designed and implemented have not been studied and clarified. In the rest of this chapter, some thoughts on designing an accurate, real-time, and flexible DIC system based on the ideas of the paDIC and paDVC algorithms are presented. The computational performance of FFT-CC and IC-GN on the multi-core CPU and GPU is carefully examined, indicating that CPU is superior to GPU in small-scale measurement while GPU is preferred when the measurement scale increases. Guided by the obtained results, a pipelined DIC system framework unifying the computation strength of both the CPU and the GPU is proposed. While the first two variations of the framework can be potentially applied to study dynamic phenomena or resolve large deformations, a third variation that may be possibly applied to ultra-fast deformation monitoring is also provided. The paDIC algorithm is even deployed to the CPU and the GPU of an iPhone 5s, further implying its potential to be applied to realize a real-time DIC system on portable devices.

5.2 Real-time DIC system framework

As a high processing speed is crucial for a real-time DIC system, the computational performance of the two employed DIC algorithms, FFT-CC and IC-GN, is first carefully compared and analyzed. The two algorithms are implemented and optimized on a multi-core CPU and a GPU. Guided by the comparison results, a hybrid CPU and GPU real-time DIC system framework is then demonstrated.

5.2.1 Analysis of computational performance of FFT-CC and IC-GN on the CPU and the GPU

While details of the GPU-based FFT-CC and IC-GN can be referred to Sections 3.4 and 4.3, the CPU multi-core implementation of them is carried out by uniformly distributing the calculations of FF-CC and IC-GN at individual POIs to multiple CPU cores. The procedure shown in Fig. 3.1 are then performed within each CPU core to obtain the final results. Note that, the precomputation phase is not considered in this test, since it is only required to be calculated once and its running time is negligible compared with the more time-consuming FFT-CC and IC-GN algorithm (see Table 3.1).

The experiment is performed on the same twenty-one frames of simulated speckle images as used in Section 3.5. The program is written using C++ with the Microsoft Visual C++ 2013 x64 compiler and the CUDA 8.0 SDK, and tested on a desktop computer equipped with an Intel Xeon CPU E5-1650 (6-core, 3.20 GHz, 16.0 GB RAM) and an NVIDIA GeForce GTX 680 GPU (8 SMs with 1536 CUDA cores and 2GB 256-bit RAM). On the CPU side, OpenMP [95] is used as the threading library, and the SSE3 intrinsic set is enabled.

Figure 5.1 illustrates the comparison of the frame rates among the implementations on 2 cores of the CPU, 6 cores of the CPU, and the GPU of the FFT-CC and IC-GN algorithms based on various number of POIs contained in each frame of image. The POI-centric subset size is set as 21×21 pixels.

(a)

(b)

Figure 5.1: Comparison of the frame rate per second (fps) of (a) the FFT-CC and (b) the IC-GN algorithms implemented by the CPU using 2 cores and 6 cores, and the GPU on different number of POIs.

It can be observed that FFT-CC is much faster than IC-GN, and its GPU implementation achieves a real-time rate (30 fps) even when 40,000 POIs are pro-

cessed each frame. However, the GPU does not show prominent advantage over the CPU when the number of POIs is less than 36. Specifically, when the number of POIs is less than 16, the CPU multi-core implementation is much faster than of the GPU. On the contrary, however, as shown in Fig. 5.1 (b), the GPU is consistently superior to the CPU for IC-GN, since IC-GN include much more complex operations that can be parallelized than those in FFT-CC. Also, it is found that the performance of the GPU varies little when the number of POIs is less than 100, which indicates that the workload allocated to the GPU is not enough to alleviate its own high latency when the problem size is small. As the number of POIs grows, the GPU continues to demonstrate its increasing superiority to the CPU.

5.2.2 Three variations of the real-time DIC system framework

A real-time DIC system requires that the processing rate should match the data acquisition rate (i.e. capturing rate), and the results should be visualized and saved to external storage without delay so that they can be directly used for further analysis. To fulfill these requirements, a pipelined system framework [89] which can overlap data acquisition and processing is considered a strong candidate.

One possible design of a general pipelined DIC system framework is shown in Fig. 5.2, in which Thread 0–2 are referred to the CPU threads. Thread 0 performs as the "master thread". While it takes charge of the response from Graphical User Interface (GUI) and result display, it is also responsible for the allocation and scheduling of other worker threads, i.e. Threads 1 and 2. In this framework, processes within each thread depend on each other, however, they can be run concurrently. For example, as soon as Thread 2 obtained the calculated results from FFT-CC and IC-GN, it begins to output the results to hard-disk while the calculation of the next frame is immediately started without affecting the output process. It is worth mentioning that, the black box containing the "FFT-CC + IC-GN" shown in Fig. 5.2 is not assigned to any CPU thread yet. To realize an accurate, real-time, and flexible DIC system, three variations on this black box is provided as follows.

Figure 5.2: Schematic of a general pipelined DIC system framework.

5.2.2.1 Variation 1: small-scale per-frame DIC system

A small-scale deformation monitoring system can be used to monitor the in situ deformation of small parts on an objects surface [21] or develop a video-based extensometer [271]. In this case, the number of POIs involved in the calculation is small. The design of the system framework shown in Fig. 5.3 can be applied, where FFT-CC is performed in multiple CPU threads while the subsequent IC-GN is offloaded to the GPU side, since as shown in Fig. 5.1(a), the CPU is superior to the GPU on FFT-CC when the number of POIs is small.

As illustrated in Fig. 5.3, on the CPU side, several CPU threads are spawned to perform FFT-CC together, and the results are passed to Thread 3 which is responsible for the communication between the CPU and the GPU as well as launching the GPU functions to perform IC-GN. It is also worth mentioning that, thanks to the CUDA and OpenGL interoperability [272], the results calculated from IC-GN on the GPU can be directly mapped to OpenGL buffers and rendered as OpenGL textures [102], eliminating the memory transferring latency between the CPU and the GPU and thus further improving the computation efficiency.

To use this framework, the running time between the CPU and GPU implementations of FFT-CC should be examined first to know when the CPU is faster than the GPU so that a proper scheduling of the CPU threads and the GPU can be obtained to reach the desired speed performance. As shown in Section 5.2.1, for example, in the proposed test, the CPU is preferred to the GPU when the number of POIs is less than 64.

Figure 5.3: Schematic of the small-scale per-frame DIC system framework.

Figure 5.4: Schematic of the large-scale per-frame DIC system framework.

5.2.2.2 Variation 2: large-scale per-frame DIC system

The direct integration of the paDIC algorithm into the proposed framework, as shown in Fig. 5.4, can be used to implement a more accurate DIC system if sub-pixel accuracy is necessary for every frame. Although this design may be slower than the one shown in Fig. 5.3, a real-time frame rates (30+ fps) can be reached

Figure 5.5: Schematic of the DIC system framework for ultra-fast deformation monitoring.

when up to 10,000 POIs are processed within each image frame.

Moreover, this design is especially useful to resolve the large deformation using an incremental method similar to [270] and [55], in which the reference image is kept updated to make the inter-frame deformation small enough to be accurately calculated. In practice, in order to achieve the optimal accuracy and maximally avoid decorrelation, the reference image is recommended to be updated every frame.

5.2.2.3 Variation 3: DIC system for ultra-fast deformation monitoring with sub-pixel compensation

A large-scale deformation monitoring system can be used to monitor deformation on the surface of large structures such as bridges [273], which requires real-time computation and visualization of displacement or strain fields within a ROI. As the measurement scale increases, running FFT-CC on the CPU is no longer an efficient solution. A system framework shown in Fig. 5.5 demonstrates a possible solution for the large-scale deformation monitoring, in which FFT-CC is performed on the GPU and the obtained integer-pixel level results are directly used for display. IC-GN, however, is executed in several CPU threads in background. And it will only

be executed as long as sub-pixel accuracy is necessary.

The design philosophy behind this framework originates from the following three aspects: (i) deformation of large-scale structures is always small compared with the distance from the cameras. However, these structures may deform very rapidly, for example, a bridge may vibrate intensively when a heavily loaded truck is passing on the bridge. Thus, an ultra-high-speed process is required to capture and visualize these rapid deformations; (ii) to achieve the required speed, parallel computing should be employed. As shown in Fig. 5.1(a), the GPU accelerated FFT-CC algorithm is a good candidate, since it is extremely fast even when tens of thousands of POIs are contained in each image frame (the processing rate of images containing 40,000 POIs is 49 fps); (iii) to further increase the computation speed, when the deformation is small, IC-GN is not required to be performed every frame. Instead, it can be performed in background and its results are only used as an compensation whenever the measurement accuracy degrade to a certain level. However, how many times the IC-GN is performed should be carefully handled, which is detailed in Section 5.2.3.

Figure 5.6: Schematic of the reference updating scheme where the reference image is updated every m frames.

5.2.3 Reference updating scheme

Fast image acquisition and processing rate creates an issue of decorrelation when the scale deformation keeps growing while the reference image remains unchanged. Thus, a reference image updating scheme is proposed in this section.

As shown in Fig. 5.6, the reference image can be straightforwardly updated every m frames. While $m = 1$ can be simply used in the first two variations of the DIC system framework, special care should be taken in choosing an appropriate m for the third variation.

In Variation 3, the integer-pixel results calculated from the GPU-based FFT-CC algorithm should be transformed back to the CPU side and saved into a concurrent buffer before they can be further refined by the CPU-based IC-GN algorithm. Due to the fact that the GPU-based FFT-CC algorithm is much faster than the CPU-based IC-GN, the size of the concurrent buffer used to cache the integer-pixel results may keep increasing and an overflow will happen if m is not carefully tuned. Thus, m is recommended to be set as the ratio between the running time of the GPU-based FFT-CC and the CPU-based IC-GN in one frame. Otherwise, either a frame dropping mechanism or temporarily writing the results to hard-disk can be considered. However, both the two solutions suffer from either an accuracy loss or a decrease in computational efficiency.

Alternatively, the reference image can be updated more wisely by frequently examining the C_{ZNCC} coefficients at each POI. In the proposed system, if the C_{ZNCC} becomes unreliable, i.e. it is less than 0.8, the reference image is updated to the newly captured frame. Moreover, to avoid error accumulation that may occur during the reference updating procedure, IC-GN is performed on the old and the updated reference images.

5.3 Example applications

Based on the three frameworks proposed in Section 3.2, a flexible real-time 2D DIC system is implemented on a PC. Figure 5.7 shows the parameter configuration dialog of the proposed system. After selecting a ROI (shown by the blue rectangle), end-users can decide which variation of the framework they want to use from the three options shown in the right panel. For example, if the "Int-pixel with sub-pixel compensation" option is selected, the third variation is enabled, and the number

of CPU threads involved in the IC-GN calculation in the background is set as the number of physical cores of the employed CPU by default.

Figure 5.7: Parameters setting dialog of the proposed real-time DIC system, where the three variations of the framework can be selected in the right panel.

In this test, the camera is fixed at 30 fps. A horizontal force is consistently exerted on the specimen, and a ROI containing 23,001 POIs is selected. Based on these configurations, the large-scale per-frame DIC framework (i.e. the "Int-pixel + sub-pixel" option in Fig. 5.7) is chosen. It is worth mentioning that, the average processing rate is also 30 fps, which matches the camera's acquisition rate. The horizontal and vertical displacement fields of the POIs at frames 0, 24, 40, and 101 are shown in Fig. 5.8.

As shown in Fig. 5.9, paDIC algorithm has also been ported to IOS devices, and an application named iDIC which was implemented on the multi-core

Figure 5.8: Visualization of the horizontal and vertical displacement fields calculated using the proposed real-time DIC system: (a-d) represent the horizontal and vertical displacement fields of POIs at frames 0, 24, 40, and 101, respectively

CPUs equipped on modern IOS devices has been published to Apple Store. While the same multi-core implementation introduced in Section 5.2.1 was employed, pThread [274] is used as the threading library since it is inherently supported by Unix-based operation systems. However, iDIC currently can only perform static DIC processing since the computation power of mobile CPU is insufficient to achieve a high processing rate even its multiple cores are enabled.

Figure 5.9: The screen-shots of the iDIC application operated on an iPhone 5s.

To further improve the processing speed of iDIC, the possibility of using the GPU acceleration in an IOS device has also been attempted. Recently, we have

Table 5.1: Comparison of the running time of the GPU-accelerated and the CPU sequential implementations of paDIC performed on different numbers of POIs on an iPhone 5s.

Number of POIs	GPU (s)	CPU(s)	Speedup ratio
1869	42.90	1.39	30.8
840	20.09	0.64	31.4
495	12.41	0.40	31
324	8.62	0.27	31.9
210	6.17	0.19	32.4

successfully ported the paDIC algorithm to the GPU of an IOS device with the help of the general-purpose GPU (GPGPU) library Metal released by Apple. The application is tested on an iPhone 5s (A7-64-bit dual-core CPU, 1.3 GHz, 1GB RAM). The resulted computation performance is compared with the sequential implementation of the same algorithm in Table 5.1, in which an remarkable 30+ times speedup is achieved.

5.4 Conclusion

In this chapter, some thoughts on designing an accurate, fast, and flexible real-time DIC system based on the ideas of paDIC and paDVC algorithms are proposed. With the guidance of the achieved processing rates of FFT-CC and IC-GN by the CPU and GPU parallel computing, three variations of a pipelined DIC system framework are demonstrated, which offer the system potential flexibility to be used in a variety of applications. The given example applications not only illustrate that a computer-based real-time DIC system with a high frame rate is already available, but also expose the potential possibility of realizing it on portable devices. In the future work, we are going to integrate the proposed three pipelined frameworks into the later versions of iDIC.

Chapter 6

Parallel reference-based dynamic phase retrieval

In DIC or DVC, the information of deformation is carried by random speckle patterns. We have systematically demonstrated how parallel computing can be applied for the speedup of DIC and DVC methods. A real-time DIC system framework has also been presented to show how DIC can be applied to study dynamic behaviors of a phenomenon.

The dynamic analysis is also required in optical interferometry, in which the evolving field of a dynamic phenomenon is represented as a dynamic phase distribution $\varphi(x, y; \tau)$ in a fringe or speckle pattern sequence (Eq. 1.19). In this chapter, based on the least-squares with three unknowns (LS3U) algorithm, which combines a least-squares fitting algorithm for phase change estimation and a windowed Fourier filtering (WFF) algorithm for denoising, a GPU-accelerated LS3U (G-LS3U) algorithm for real-time dynamic phase retrieval, is proposed. G-LS3U achieves a 131.8 fps processing rate, which is the first real-time reference-based dynamic phase retrieval algorithm published heretofore.

6.1 Introduction

As mentioned in Section 1.4, there are three categories of methods in dynamic phase retrieval from fringe or speckle pattern sequences: phase shifting methods, transform-based methods, and reference-based methods. In phase shifting

methods [63, 64], phase distributions are extracted using the conventional phase shifting methods that developed for statical phase extraction, which are very time-consuming to be used in dynamic processes. Transform-based methods, including the Fourier transform [68–70], the Hilbert transform [71, 72], the temporal wavelet transform [73, 74], and the windowed Fourier transform (WFT) [75], can extract the phase distribution from a single fringe of speckle pattern. However, the measurement range is limited by the introduction of carriers.

Reference-based methods [76–80] only extract the phase of a reference state using phase shifting or Fourier transform. The phases of all consecutive states are determined by estimating the phase changes between the current state and the reference state. There are three main problems of the reference-based methods: (i) the speckle decorrelation problem in speckle-based interferometry [62]; (ii) the background intensity $a(x, y; \tau)$ and the amplitude $b(x, y; \tau)$ have to be assumed to be constant [79, 80] or estimated by the scanning phase method [76–78]; (iii) the error accumulation problem may occur when the final phase is calculated by accumulating phase changes.

To solve these problems, a reference-based dynamic phase retrieval algorithm, called LS3U, was proposed [61, 62, 81, 275]. In LS3U, the phase changes as well as the background intensity and amplitude (Eq. 1.19) were estimated by a least-squares fitting algorithm, and then denoised by the WFF algorithm [61]. The final phase is obtained by accumulating the denoised phase changes. LS3U is superior to other existing dynamic phase retrieval algorithms in five aspects: (i) no high-speed phase-shifting devices are required, since phase shifting is only required to be performed for the initial state; (ii) the measurement range is not limited, since no temporal or spatial carrier is required; (iii) the background intensity and amplitude do not have to be estimated in advance or assumed to be constant since they are estimated together with the phase changes; (iv) a re-referencing scheme is proposed to avoid speckle decorrelation; (v) the error accumulation is eliminated by the effective WFF. All these advantages offer LS3U a potential to be widely applied as a fast and robust dynamic phase retrieval algorithm.

The computation burden of LS3U, however, is heavy due to the per-pixel least-squares fitting and the WFF algorithm. The conventional MATLAB implementation of LS3U requires 1.25s to process a 256×256 image [275]; the frame rate is thus 0.8 fps, which is much lower than a video rate of 24fps or 30fps. It indicates that the phase retrieval from a fast dynamic phenomenon should be performed off-line. In fact, there is a high parallelism in LS3U and it should be very suitable for being accelerated by parallel computing. As illustrated in Section 6.3.1, the least-square per-pixel least-squares fitting can be well parallelized using the pixelwise pattern. Furthermore, parallel WFF algorithms have also been attempted in several research works [139, 174, 276–278], among which the GPU-based WFF algorithm has achieved a $132\times$ speedup than its serial implementation [277].

In this chapter, a real-time LS3U algorithm, G-LS3U, powered by the GPU implementing using CUDA [90] is proposed. The computation efficiency of G-LS3U is verified using two sequences of fringe and speckle patterns recorded from real experiments. Real-time processing rates of up to 64.5fps and 131.8fps are achieved on NVIDIA’S GTX 680 and GTX 1080 graphics cards, respectively. G-LS3U is the first real-time reference-based dynamic phase retrieval algorithm.

6.2 Principle of LS3U

In LS3U, the phase distribution $\varphi(x, y; 0)$ (Eq. 1.18) is estimated by the advanced iterative algorithm (AIA) [82] and used as the initial reference phase. The phases of the consecutive frames are then extracted by LS3U in two stages: a least-squares fitting stage to estimate the phase changes, and a WFF denoising stage to refine the phase changes for the phase update.

6.2.1 Least-squares fitting for phase change estimation

A small neighborhood of a pixel (u, v) in the $f(x, y; \tau)$ defined in Eq. 1.19 is defined as $NB(u, v) = (x, y) | \max(|x - u|, |y - v|) \leq \epsilon$, where ϵ is a small positive integer. In LS3U, based on the spatial continuity, the $a(x, y; \tau)$, $b(x, y; \tau)$, and $\Delta\varphi(x, y; \tau_0, \tau)$

are assumed to be constant in the $NB(u, v)$, i.e., $a(x, y; \tau) \approx a(u, v; \tau)$, $b(x, y; \tau) \approx b(u, v; \tau)$, and $\Delta\varphi(x, y; \tau_0, \tau) \approx \Delta\varphi(u, v; \tau_0, \tau)$. With these assumptions, for each $(x, y) \in NB(u, v)$, the $f(x, y; \tau)$ in Eq. 1.19 can be re-written as

$$\begin{aligned} f(x, y; \tau) &\approx a(u, v; \tau) + b(u, v; \tau) \cos [\bar{\varphi}(x, y; \tau_0) + \Delta\varphi(u, v; \tau_0, \tau)] \\ &= a(u, v; \tau) + c(u, v; \tau_0, \tau) \cos [\bar{\varphi}(x, y; \tau_0)] + d(u, v; \tau_0, \tau) \sin [\bar{\varphi}(x, y; \tau_0)], \end{aligned} \quad (6.1)$$

where

$$c(u, v; \tau_0, \tau) = b(u, v; \tau) \cos [\Delta\varphi(u, v; \tau_0, \tau)], \quad (6.2)$$

$$d(u, v; \tau_0, \tau) = -b(u, v; \tau) \sin [\Delta\varphi(u, v; \tau_0, \tau)]. \quad (6.3)$$

There are M equations of Eq. 6.1 in total, where M is the number of pixels in the $NB(u, v)$. The M equations are linear based on three unknowns, i.e., $\mathbf{p}(u, v; \tau) = [a(u, v; \tau), c(u, v; \tau), d(u, v; \tau)]^T$ written in a vector form. If $M \geq 3$, i.e., $\epsilon \geq 1$, the unknowns can be solved in a least-squares manner as [81]

$$\mathbf{A}\hat{\mathbf{p}}(u, v; \tau) = \mathbf{B}, \quad (6.4)$$

where

$$\mathbf{A} = \begin{bmatrix} M & \sum_{x,y} \cos[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} \sin[\bar{\varphi}(x, y; \tau_0)] \\ \sum_{x,y} \cos[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} \cos^2[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} \cos[\bar{\varphi}(x, y; \tau_0)] \sin[\bar{\varphi}(x, y; \tau_0)] \\ \sum_{x,y} \sin[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} \cos[\bar{\varphi}(x, y; \tau_0)] \sin[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} \sin^2[\bar{\varphi}(x, y; \tau_0)] \end{bmatrix}, \quad (6.5)$$

and

$$\mathbf{B} = \begin{bmatrix} \sum_{x,y} f(x, y; \tau_0) & \sum_{x,y} f(x, y; \tau_0) \cos[\bar{\varphi}(x, y; \tau_0)] & \sum_{x,y} f(x, y; \tau_0) \sin[\bar{\varphi}(x, y; \tau_0)] \end{bmatrix}^T, \quad (6.6)$$

where the $\sum_{x,y}$ represents the summation for the $(x, y) \in NB(u, v)$; the $\hat{\mathbf{p}}(u, v; \tau)$ is obtained as an estimation of the $\mathbf{p}(u, v; \tau)$ due to noise. The phase changes can then be estimated as

$$\widehat{\Delta\varphi}(u, v; \tau_0, \tau) = -\arctan \left[\frac{\hat{d}(u, v; \tau_0, \tau)}{\hat{c}(u, v; \tau_0, \tau)} \right]. \quad (6.7)$$

6.2.2 WFF for noise elimination

The phase change estimation $\widehat{\Delta\varphi}(u, v; \tau_0, \tau)$ is always noisy. To avoid error accumulation of the phase update, the highly effective WFF algorithm [61, 62] is employed for noise elimination. As in [62], an estimated phase change is first converted to an exponential phase field (EPF) as

$$\Phi(u, v; \tau_0, \tau) = \exp \left[j \widehat{\Delta\varphi}(u, v; \tau_0, \tau) \right], \quad (6.8)$$

where $j = \sqrt{-1}$. The $\Phi(u, v; \tau_0, \tau)$ is then transformed to the windowed Fourier space by WFT and filtered by the WFF algorithm. Afterwards, the filtered EPF $\Phi(u, v; \tau_0, \tau)$ is obtained from an inverse WFT. Finally, the filtered phase change $\overline{\Delta\varphi}(u, v; \tau_0, \tau)$ is calculated as

$$\overline{\Delta\varphi}(u, v; \tau_0, \tau) = \text{angle}[\Phi(u, v; \tau_0, \tau)]. \quad (6.9)$$

The current phase $\varphi(u, v; \tau)$ can thus be updated by substituting Eq. 6.9 to Eq. 1.18, which is denoted as $\bar{\varphi}(u, v; \tau)$ to highlight that it is an estimation obtained from the least-squares fitting and WFF filtering processes.

6.2.3 LS3U algorithm

The sequential LS3U algorithm is summarized in Algorithms 6.1 and 6.2. In LS3U, $\epsilon = 1$ is used so that $M = 9$ for inner pixels ($M = 6$ and $M = 4$ for boundary and corner pixels, respectively) in the $NB(u, v)$. This configuration best fulfills the assumption of constant background intensity and amplitude and it is sufficient to solve the 3 unknowns. Moreover, the re-referencing rate is set as most frequent as $rr = \tau - \tau_0 = 1$ [62], which best satisfies the assumption of constant phase changes [81]. Thus, the speckle decorrelation problem in speckle-based interferometry is automatically resolved. Since WFF was proposed more than a decade ago [61] and its GPU implementation has also been attempted [277], details of the GPU-based WFF are thus omitted. However, some implementation tips that make WFF be more efficient in dynamic processes are introduced in Section 6.3.2.

Algorithm 6.1 Sequential LS3U algorithm.

Initialization: $\epsilon \leftarrow 1$, $M \leftarrow 9$, $rr \leftarrow 1$;
for each frame τ in a sequence **do**
 $f_\tau = \text{loadImage}(\tau)$;
 $\bar{\varphi}(u, v; \tau) = \text{LS3U_per_frame}(f_\tau)$; // See Algorithm 6.2.
 if $\tau \% rr == 0$ **then**
 Update the reference phase map: $\bar{\varphi}(u, v; \tau_0) \leftarrow \bar{\varphi}(u, v; \tau)$;
 end if
end for

Algorithm 6.2 Sequential LS3U_per_frame algorithm.

for each pixel (u, v) in an image frame f_τ **do**
 1. Construct **A** and **B**; // Eqs. 6.5 and 6.6
 2. Solve $\hat{\mathbf{p}}(u, v; \tau)$; // Eq. 6.4
 3. Compute $\Delta\bar{\varphi}(u, v; \tau_0, \tau)$ and $\Phi(u, v; \tau_0, \tau)$ // Eqs. 6.7 and 6.8
end for
 4. $\hat{\Phi}(u, v; \tau_0, \tau) = \text{WFF}(\Phi)$;
 5. Compute $\Delta\bar{\varphi}(u, v; \tau_0, \tau)$ and $\bar{\varphi}(u, v; \tau)$; // Eqs. 6.9 and 1.18

6.3 GPU-powered real-time LS3U

As introduced in Section 6.2, LS3U obtains its five advantages listed in Section 6.1 by using the intelligent yet complicated algorithms. These algorithms bring a heavy computation burden to LS3U. Dynamic phase retrieval using the naïve serial implementation of LS3U shown in Algorithms 6.1 and 6.2 thus can only be performed off-line.

To reach a higher or even real-time data processing rate, various parallel computing hardware, including multi-core CPUs, FPGAs, GPUs, etc., have been employed to improve the computation performance of optical measurement techniques [89], among which CPUs and GPUs are the most widely used [83] (see Chapter 2). The proposed G-LS3U is implemented on the GPU using CUDA due to its easy programming interface and highly optimized parallel computation libraries.

As shown in Algorithms 6.1 and 6.2, there are two loops involved in the LS3U algorithm: a framewise loop among the captured sequence of image frames (Algorithm 6.1) and a pixelwise loop among all the pixels within one image frame.

Figure 6.1: Flowchart of the G-LS3U algorithm. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

As shown in Fig. 6.1, the framewise loop remains serial in the proposed G-LS3U, since these image frames are captured in chronological order in real-time applications. The pixelwise loop, however, can be well parallelized. Algorithm 6.3 shows the parallelization of Algorithm 6.2, which uses the pixelwise pattern, the tiling pattern, and the divide-and-conquer pattern introduced in Section 2.2. Steps 2, 3, and 5 follows the pointwise pattern. Step 1 contains a tiling pattern, where the construction of $\mathbf{A}$ and $\mathbf{B}$ of one pixel should involve its M neighbors. The FFTs used in the WFF algorithm in Step 5 is a divide-and-conquer pattern. Details of the parallel computation strategies applied to the least-squares fitting and the WFF algorithms are described as below.

Algorithm 6.3 Parallelized LS3U_per_frame algorithm.

```
// I/O from host memory to device memory
0. Load  $f_\tau$  to GPU Device memory  $\mathbf{d\_img}$ ;

// Use least-squares fitting method to obtain the phase changes
1. a.  $[\mathbf{d\_cos}\bar{\varphi}, \mathbf{d\_sin}\bar{\varphi}] = \text{cosPhi\_sinPhi\_kernel}(\mathbf{d\_}\bar{\varphi})$ ;
   b. Construct  $\mathbf{d\_A}$  and  $\mathbf{d\_B}$  // Eqs. 6.5 and 6.6
       $[\mathbf{d\_A}, \mathbf{d\_B}] = \text{construct\_A\_B\_tile\_kernel}(\mathbf{d\_cos}\bar{\varphi}, \mathbf{d\_sin}\bar{\varphi})$ ;
2. Solve  $\mathbf{d\_}\hat{\mathbf{p}}$  //Eq. 6.4
    $\mathbf{d\_}\hat{\mathbf{p}} = \text{Gaussian\_Elimination\_3x3\_kernel}(\mathbf{d\_A}, \mathbf{d\_B})$ ;
3. Compute  $\mathbf{d\_}\overline{\Delta\varphi}$  and  $\mathbf{d\_}\Phi$  // Eqs. 6.7 and 6.8
    $\mathbf{d\_}\Phi = \text{deltaPhi\_and\_EPF\_kernel}(\mathbf{d\_}\hat{\mathbf{p}})$ ;

// Use the parallel WFF algorithm to suppress noise
4.  $\mathbf{d\_}\hat{\Phi} = \text{paWFF}(\mathbf{d\_}\Phi)$ ;
5. Compute  $\mathbf{d\_}\overline{\Delta\varphi}$  and  $\mathbf{d\_}\bar{\varphi}$  // Eqs. 6.9 and 1.18
    $[\mathbf{d\_}\overline{\Delta\varphi}, \mathbf{d\_}\bar{\varphi}] = \text{filtered\_deltaPhi\_and\_Phi\_kernel}(\mathbf{d\_}\hat{\Phi})$ ;

// I/O from Device memory to Host memory
6. Copy  $\mathbf{d\_}\overline{\Delta\varphi}$  and  $\mathbf{d\_}\bar{\varphi}$  to  $\mathbf{h\_}\overline{\Delta\varphi}$  and  $\mathbf{h\_}\bar{\varphi}$ ;
```

6.3.1 Parallel computing strategies applied to the least-squares fitting algorithm

For clarity, prefixes $h_$ and $d_$ are used to indicate variables on the CPU (Host) and the GPU (Device), respectively.

6.3.1.1 Data transfer

Before launching GPU routines, the required data should be first transferred to the GPU's global memory. After the computation, the results should be copied back to the CPU side. Compared with Algorithm 6.2, Steps 0 and 6 in Algorithm 6.3 are the extra memory copying operations between the host (CPU) and the device (GPU) memory. Thanks to the pinned host memory [90], as shown in Table 6.2 in Section 6.4, the memory transferring time is negligible compared to the other operations.

6.3.1.2 Parallelization of the pointwise pattern

The per-pixel least-squares fitting algorithm is very time-consuming when the image size is large. However, as shown in Eqs. 6.4, 6.7, and 6.8, each pixel can be independently processed, which well fits the pixelwise pattern and thus is suitable to be accelerated on the GPU. Steps 2, 3, and 5 in Algorithm 6.2 are thus parallelized using the pixelwise pattern. Their parallelized counterparts are shown in Algorithm 6.3. Step 3, for example, allocates CUDA threads equal to the number of pixels of an image frame. Afterwards, each CUDA thread uses the same instructions as in Eqs. 6.7 and 6.8 to calculate the per-pixel $\widehat{\Delta\varphi}(u, v; \tau_0, \tau)$ and $\Phi(u, v; \tau_0, \tau)$.

6.3.1.3 Parallelization of the tiling pattern

The construction of the $\mathbf{A}$ and $\mathbf{B}$ shown in Step 1b in Algorithm 6.3 at each pixel (u, v) requires to access its M neighbors, which can be more efficiently parallelized by the tiling pattern. Instead of fetching the neighboring positions of each pixel from the global memory redundantly, all the associated neighbors of a small group

of pixels are cached into the GPU's fast on-board shared memory, which is more than $10\times$ faster than the global memory. All the subsequent processing of each small group thus fetches the required data from the corresponding shared memory positions. The neighbor association, called tiling, is simple. An original image is first uniformly divided into smaller sub-images, each of which is called a tile and $K \times K$ pixels. Several extra pixels should be included to deal with the calculation at boundary pixels, i.e., each tile is expanded to the size of $K_e \times K_e$ with $K_e = K + \epsilon$ where ϵ is the parameter used to define the $NB(u, v)$ in Section 6.2. CUDA blocks with $K_e = K + \epsilon$ threads are allocated to process the tiles of $K \times K$ pixels. All the $K_e \times K_e$ threads within a block participate in caching the $K_e \times K_e$ pixels into the block's shared memory, while only the $K \times K$ threads are responsible for the subsequent construction of the **A** and **B**. As CUDA prefers a block size of a power of 2 and the maximum allowed block size is 1024 [90], $K = 14$ and $K_e = 16$ are used in the proposed implementation to maximally utilize the GPU's resources.

Texture memory, as an alternative, can also be used for cached neighboring fetches. But it is not considered in G-LS3U due to the following two aspects: (i) the allocation of texture is opaque, which makes it less flexible than the standard CUDA memory; (ii) texture originates from computer graphics applications of the GPU. It is read-only and has its own application programming interfaces (APIs). Especially, on the GPU with computation capability less than 2.0, these APIs cannot be invoked by CUDA kernels directly, thus restricting the portability and flexibility of G-LS3U. Since the computation speed achieved by the tiling method is remarkable (Table 6.2), it is thus chosen as the solution.

6.3.1.4 Reduce redundant computations

In the construction of the **A** and **B** using the tiling pattern, as shown in Eqs. 6.5 and 6.6, $\cos[\bar{\varphi}(u, v; \tau_0)]$ and $\sin[\bar{\varphi}(u, v; \tau_0)]$ should be used multiple times at pixels $(u, v; \tau_0)$, $(u, v + 1; \tau_0)$, $(u + 1, v; \tau_0)$, etc. Unnecessary redundancy will result if these quantities are calculated whenever they are required. Thus, in G-LS3U, $\cos[\bar{\varphi}(u, v; \tau_0)]$ and $\sin[\bar{\varphi}(u, v; \tau_0)]$ of an entire image frame are precomputed in

Step 1a and saved into two GPU arrays, i.e., $d_cos\bar{\varphi}$ and $d_sin\bar{\varphi}$ residing in global memory.

6.3.1.5 Equation solving

In Step 2 of Algorithm 6.3, each thread is required to solve a 3×3 linear system (Eq. 6.4). Cholesky and lower-upper (LU) factorization are two possible efficient solutions since $\mathbf{A}$ is symmetric. However, the positive definiteness of $\mathbf{A}$ cannot be guaranteed so that the faster Cholesky method cannot be applied. Also, LU and the Gaussian elimination algorithm have the same time complexity of $O(n^3)$ in solving a linear system with only one right-hand side. Therefore, the simpler Gaussian elimination is employed in G-LS3U. Moreover, the Gaussian elimination with partial pivoting (GEPR) algorithm is implemented to obtain more robust and stable results. In G-LS3U, GEPR is used to factorize the per-thread 3×4 augmented matrix, and the final results are obtained by a back substitution [280].

6.3.2 Parallel computing strategies applied to the WFF algorithm

The parallel WFF algorithm (paWFF) used in Step 4 in Algorithm 6.3 has already been proposed [277]. However, to make it be more efficient in a real-time dynamic system, two precomputation tips are proposed in G-LS3U, which are the precomputation of the FFT of the windowed Fourier basis function and the the precomputation of the LUT for FFT preferred sizes. With these two tips, about a quarter of the WFF workload can be removed by precomputation.

6.3.2.1 Precomputation of FFT of the windowed Fourier basis function

In WFF, FFT is performed on the input pattern $f(x, y; \tau)$ and the windowed Fourier basis function g_{ξ_x, η_y} . While the former is calculated as $Ff(\xi, \eta; \tau)$ using cuFFT, the latter can be obtained analytically as

$$Fg_{\xi_x, \eta_y}(\xi, \eta) = \sqrt{4\pi\sigma_x\sigma_y} \exp \left[-\frac{\sigma_x^2(\xi - \xi_x)^2 + \sigma_y^2(\eta - \eta_y)^2}{2} \right], \quad \xi \in [\xi_l, \xi_h], \eta \in [\eta_l, \eta_h] \quad (6.10)$$

where σ_x and σ_y control the window size; ξ_l , x_{i_h} , η_l , and η_h control the frequency band for filtering; and $Fg_{\xi_x, \eta_y}(\xi, \eta)$ is the Fourier transform of the g_{ξ_x, η_y} . It is recommended in [277] that Eq. 6.10 can be more efficiently parallelized on the GPU using the pixelwise pattern.

In the proposed general-purpose paWFF [277], all the parameters, i.e., σ_x , σ_y , ξ_l , x_{i_h} , η_l , and η_h should be set by a user. On the contrary, in G-LS3U, these parameters are selected before an experiment starts and remain invariant during the entire experiment. The $Fg_{\xi_x, \eta_y}(\xi, \eta)$ thus can be precomputed and stored in global memory.

Figure 6.2: Schematic of the shifting process.

However, special care is needed in precomputing $Fg_{\xi_x, \eta_y}(\xi, \eta)$ since in the (ξ, η) plane of $Fg_{\xi_x, \eta_y}(\xi, \eta)$, the DC component is at the center, while it is at the upper-left corner of the $Ff(\xi, \eta; \tau)$. Therefore, a shifting process as shown in Fig. 6.2 should be used to align them. In G-LS3U, the $Fg_{\xi_x, \eta_y}(\xi, \eta)$ is shifted to match $Ff(\xi, \eta; \tau)$, since the shifting can also be precomputed. A parallel shifting algorithm called *cufftshift* implemented on the GPU has been proposed [281], but it is only applicable to square images. In G-LS3U, we extend the capability of *cufftshift* to make it applicable to general rectangle images yet without losing computation efficiency.

6.3.2.2 Precomputation of the Look-up table for FFT preferred sizes

In the conventional WFF algorithm [277], to achieve a faster computation speed using *cuFFT*, an image was expanded to the nearest power of a factor t . However,

this configuration traded excessive memory consumption for speed. Indeed, as described in [110], cuFFT has already been optimized for the sizes in the form of $S = 2^a \times 3^b \times 5^c \times 7^d$ ($a, b, c, d = 0, 1, 2, \dots$). To determine a , b , c , and d , a pre-factorization method [139] has to be performed whenever the size of an image changes. In G-LS3U, a more efficient LUT containing preferred S 's is precomputed and saved in a constant 1D sequence. Moreover, to keep a reasonable size of the LUT and make the shifting process mentioned above more convenient, the S should satisfy the following two criteria:

- (i) $S \% 2 = 0$, where $\%$ indicates the modular operator;
- (ii) $S < 5000$.

6.4 Experimental verification

The proposed G-LS3U is implemented using Visual C++ 2013 and CUDA 8.0. To make the comparison fair, in addition to the conventional MATLAB implementation [81], a CPU multi-core accelerated LS3U is also provided with four optimizations: (i) OpenMP [95] is employed to parallelize the loop in Algorithm 6.2; (ii) The LAPACK routine `zsysv`, optimized by the Intel's Math Kernel Library (MKL) [106] is used to solve the linear systems in Eq. 6.4; (iii) the recommendations provided in [139] is employed to implement the multi-core WFF algorithm; (iii) and the SSE3 is enabled in compilation. All the tests in this section are performed on a Dell Precision workstation equipped with an Intel Xeon E5-1650 CPU (6 cores, 3.20 GHz, and 16GB RAM). Both a low-end GTX 680 (8 SMs, 1536 CUDA cores, and 2GB RAM) and a high-end GTX 1080 (8 SMs, 2560 CUDA cores, and 8GB RAM), are used to test G-LS3U. Two examples recorded from real experiments are calculated, compared, and analyzed as below.

6.4.1 Dynamic fringe projection

In the first example, motion of a piece of A4 paper is measured by a dynamic fringe projection profilometer and a sequence of 108 frames of fringe patterns is

recorded with a video camera at a frame rate of 30 fps. The size of each image frame is 256×256 [62]. As recommended in [62], the parameters used by WFF are: $\sigma_x = \sigma_y = 10$; the corresponding sampling interval are set as $\xi_i = 1/\sigma_x = 0.1$ and $\eta_i = 1/\sigma_y = 0.1$; due to the small spatial variation of the phase change, and by considering the energy leakage [62], the frequency bands are set as $\xi_l = -3\xi_i = -0.3$, $\xi_h = 3\xi_i = 0.3$, $\eta_l = -3\eta_i = -0.3$, and $\eta_h = 3\eta_i = 0.3$; and $\text{thr} = 5$. Two representative fringe pattern frames, frames 20 and 64 are shown in Figs. 6.3 (a) and 6.3 (b), and the corresponding extracted phases are shown in Figs. 6.3 (c) and 6.3 (d), respectively. Different implementations of LS3U obtain the same result.

Figure 6.3: LS3U for dynamic fringe projection profilometry: (a) and (b) fringe patterns at two representative frames; (c) and (d) the corresponding wrapped phase maps extracted by LS3U. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

Table 6.1 compares the computation speed of the MATLAB sequential LS3U, the CPU multi-core implementation of LS3U, and G-LS3U. The MATLAB has the lowest frame rate of 0.4fps (which is slower than the result in [81] due to different parameters). The CPU multi-core implementation is able to extract the phase at 10.1fps, however, it is still slower than the acquisition rate. G-LS3U on the GTX 680 and GTX 1080 achieve 64.5fps and 131.8fps, respectively, which are much higher than the acquisition rate. The GTX 1080 is 2+ times faster than the GTX 680 due to the fact that it contains a larger number of CUDA cores so that more operations can be run in parallel. The computation performance of each step of Algorithm 6.3 is further examined in Table 6.2.

Steps 0 and 6: the device and host memory transfer only occupy a very small portion of the overall running time by employing the pinned host memory [90];

Table 6.1: Comparison of the average running time and frame rates among the MATLAB, the CPU Multi-Core, and the proposed GPU-Based implementations of LS3U of the first example. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

MATLAB (s/fps)	CPU 6-Core (s/fps)	GTX 680 (s/fps)	GTX 1080 (s/fps)	Speedup Ratio	
				GTX 680 versus CPU 6-Core	GTX 1080 versus CPU 6-Core
279.1/0.4	10.7/10.1	1.7/64.5	0.8/131.8	6.4	13.0

Steps 1–3: The least-squares fitting algorithm parallelized on the GTX 1080 takes less than 1ms, which is very fast. In fact, the fast speed is mainly resulting from the parallel linear system solver used to solve Eq. 6.4. As shown in Table 6.3, compared with the CPU multi-core implementation, the parallel solver achieves speedup ratios of 135.42 and 367.57 on the GTX 680 and GTX 1080, respectively.

Step 4: The paWFF algorithm dominates the overall running time of G-LS3U, which is more than one order of magnitude than the other steps. Table 3 compares the MATLAB, CPU multi-core, and GPU implementations of the WFF algorithm, where speedup ratios of 4.77 and 11.23 are achieved on the GTX 680 and GTX 1080, respectively.

Step 5: The calculation of the phase change, similar to the memory transfer in Steps 0 and 6, only occupies a very small portion of the overall running time.

6.4.2 Dynamic speckle shearographic interferometry

In the second example, deformation of a fully clamped circular plate is measured by a dynamic speckle shearographic interferometer and a sequence of 117 frames of fringe patterns are recorded with a video camera at a frame rate of 30 fps. The size of each image frame is 366×371 [62]. As recommended in [62], the parameters used by WFF are: $\sigma_x = \sigma_y = 20$; the corresponding sampling intervals are set as $\xi_i = 1/\sigma_x = 0.05$ and $\eta_i = 1/\sigma_y = 0.105$; due to the small spatial variation of the phase change, and by considering the energy leakage [62], the frequency

Table 6.2: Average running time of each step of G-LS3U on one image frame of the first example. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

Step	GTX 680 (ms)	GTX 1080 (ms)
0	0.023	0.012
1	0.165	0.065
2	0.191	0.072
3	0.017	0.017
4	14.631	6.216
5	0.020	0.010
6	0.048	0.028

Table 6.3: Comparison of the average computation time of solving the linear system (Eq. 6.4) and WFF among the MATLAB, the CPU Multi-Core, and the proposed GPU-Based implementations on the first example. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

Operation	MATLAB (ms)	CPU 6-Core (ms)	GTX 680 (ms)	GTX 1080 (ms)	Speedup Ratio	
					GTX 680 versus CPU 6-Core	GTX 1080 versus CPU 6-Core
Solving Eq. 6.4	558.20	25.73	0.19	0.07	135.42	367.57
WFF	372.61	70.06	14.68	6.23	4.77	11.23

bands are set as $\xi_l = -3\xi_i = -0.15$, $\xi_h = 3\xi_i = 0.15$, $\eta_l = -3\eta_i = -0.15$, and $\eta_h = 3\eta_i = 0.15$; and $\text{thr} = 10$. Two representative fringe pattern frames, frames 40 and 104 are shown in Figs. 6.4 (a) and 6.4 (b), and the corresponding extracted phases are shown in Figs. 6.4 (c) and 6.4 (d), respectively. Frame rates of 31.8fps and 53.7fps are achieved on the GTX 680 and GTX 1080, respectively.

Figure 6.4: LS3U for dynamic speckle shearographic profilometry: (a) and (b) speckle correlation patterns at two representative frames; (c) and (d) the corresponding wrapped phase maps extracted by LS3U. Adapted with permission from [279], Appl. Opt. 56, 7726 (2017).

6.4.3 Discussion

As shown in Table 6.2, WFF dominates the overall running time of G-LS3U. Therefore, a proper tuning of the WFF parameters is crucial to the speed performance of G-LS3U. Generally, if no prior knowledge is known on the experiment, the parameter setting used in the second example is recommended. If the experiment is known to contain mild noise, the parameter setting in the first example is more suitable for higher acceleration. Further adjustment on the parameters can be attempted if even more knowledge of the experiment can be known.

6.5 Conclusion

A real-time G-LS3U algorithm for fast dynamic phase retrieval from fringe and speckle patterns is proposed and implemented. Real-time frame rates of up to 64.5fps and 131.8fps are achieved using the GTX 680 and GTX 1080 GPUs, respectively. The paWFF algorithm has also tested on simulated fringe patterns

containing 1024×1024 pixels using the same parameters as in Section 6.4.1. An average computation speed of 75.8ms/frame has been achieved tested on the GTX 1080. Since WFF dominates the overall running time of G-LS3U, it can be safely estimated that G-LS3U can still achieve a high frame rate of 13fps under such a high resolution. Even higher frame rates can be expected if prior knowledge is known to adjust the WFF parameters. In practice, higher-end GPUs are preferred in G-LS3U, not only for their greater computation speed, but also for their larger memory space that can hold and process larger-scale images.

Chapter 7

Conclusion and future work

7.1 Summary

Due to the advantages of non-contact, non-destructiveness, and high accuracy and sensitivity, optical measurement techniques have been successfully developed and applied to study shapes and deformations of objects. Digital image correlation (DIC) and fringe pattern analysis (FPA) are two important categories of the optical measurement techniques. In the past decade, both DIC and FPA have been well studied and developed, however, two main problems still need further investigation. First, they can be hardly applied to study dynamic phenomena or integrated into real-time systems due to the increasing computation burden caused by the pursuit of higher measurement resolution and accuracy. Second, decorrelation is a serious problem when the large deformation appears between the initial and the deformed configurations of an object. In fact, these two problems can be solved by increasing the computation efficiency of DIC and FPA. Therefore, the requirement of high frame rates in dynamic measurements can be satisfied. With a high processing speed, large deformation problem can also be resolved by inserting intermediate frames between the initial and deformed states to make the inter-frame deformation small enough to use the existing methods. In the past five years since 2012, heterogeneous parallel computing platforms composing of CPUs and Graphics Processing Units (GPUs) have been broadly adopted to accelerate optical measurement. This thesis has provided a systematic study of powering optical

measurement techniques, especially the DIC and FPA techniques, using the CPU and GPU parallel computing.

First, the CPU and GPU parallel computing that has been applied to optical measurement techniques, such as DIC and digital volume correlation (DVC), FPA, tomography, hyperspectral imaging, computer-generated hologram (CHG), and integral imaging in the past 5 years since 2012 has been reviewed and analyzed. It has been found that the four common parallel patterns, namely the pixelwise pattern, the tiling pattern, the divide-and-conquer pattern, and the rendering and interpolation pattern, can always be exploited from these methods. Readers can consider to accelerate their existing techniques using these four patterns or the reviewed methods in their future applications.

Based on the path-independent DIC (Pi-DIC) algorithm that uses a path-independent initial guess transferring scheme, a high-accuracy GPU-accelerated parallel DIC (paDIC) algorithm has been proposed, which utilizes a fast Fourier transform-based cross-correlation (FFT-CC) algorithm for integer-pixel level displacement estimation and an inverse compositional Gauss-Newton (IC-GN) algorithm for sub-pixel refinement. A 57.5+ times speedup has been achieved by paDIC compared with the serial implementation of the same algorithm yet without losing accuracy.

Subsequently, paDIC has been extended to implement a 3D parallel DVC (paDVC) algorithm, in which the 3D FFT-CC and the 3D IC-GN are employed. Although paDVC is a direct extension of paDIC, more complexities have been introduced by the additional dimension. Thus, more details and optimizations of paDVC have been explained. To efficiently process a large amount 3D data volume, a batch processing scheme has been proposed to divide the entire 3D volume into smaller data cubes which are then processed in turn on the GPU. The proposed paDVC algorithm has achieved speedup ratios of 23.3 and 3.7 compared with its CPU serial and multi-core implementations, respectively.

Based on paDIC and paDVC, a real-time DIC system framework, which combines the strength of both the CPU and the GPU has been proposed to be applied

in dynamic measurement and resolving large deformation problem. While the main logics are pipelined on the CPU, the computationally intensive tasks are off-loaded to the GPU side. With the flexible three variations of the framework, a real-time processing rate ranging from 30fps to 130fps has been achieved. Moreover, a reference updating scheme has been proposed to compensate the decorrelation problem that may appear in measuring large deformation.

Last, a real-time reference-based dynamic phase retrieval (G-LS3U) algorithm has been proposed and applied to extract phase distributions within fringe and speckle patterns with a ultra-high frame rate. Different parallel computing strategies have been employed to both the least-squares fitting and the windowed Fourier filtering (WFF) algorithms. A real-time frame rate of up to 131.8fps has been achieved by G-LS3U, which makes it the fastest reference-based dynamic phase retrieval algorithm reported heretofore.

7.2 Future work

7.2.1 Further improve the computation efficiency of DIC and DVC

It has been demonstrated that, both paDIC and the rtDIC systems based on paDIC have achieved satisfactory computation efficiency compared with the conventional DIC methods [20, 255]. It is worth noting that, however, further speed performance boost can still be expected from the following two aspects. First, the most straightforward speedup can be expected if more powerful CPUs and GPUs are used. In reality, the acceleration by combining the computation powers of multiple computation nodes composed of multiple CPUs and GPUs have been attempted in [35]. Thus, in the near future, we will try to port the proposed rtDIC system framework to computation nodes to achieve an even higher frame rate. However, the high-latency of communications among these nodes may be a challenging problem that should be carefully resolved.

Second, paDIC and paDVC have much room for improvement. As shown in Fig. 1.4, a large portion of a PIO-centric subset indeed have a large portion that overlap

with its neighboring subsets. As advocated in [52], the same quantities at these overlapped regions remain the same and only needed to be calculated once. As more than 75% overlap may appear for a dense DIC or DVC analysis, eliminating these redundant calculations at each subset will certainly bring remarkable speedup to paDIC and paDVC. We are thus going to refine our paDIC and paDVC algorithms by considering the overlapping regions.

7.2.2 Resolve deformation at boundaries and edges

The displacement maps of Fig. 3.5 obtained by the paDIC are shown in Fig. 3.7. It can be observed that, the displacement fields around the hole in the center are not accurately calculated, since some portion of the POIs may fall in the hole and thus lose precision.

Recently, Zhu et al. proposed a non-central subset-based DIC method [282], which effectively circumvented the computation issue along the boundary of in a ROI. They proved that a POI does not have to be located in the center of a subset, and the position of a POI within a subset exerts trivial influence on the overall accuracy of the measurement. They then applied their new discovery to measure the boundary deformation. The subsets involved in the non-central subset-based DIC method were first defined the same as a conventional DIC method, however, an additional judgment of whether a POI is within the ROI or not was performed before solving the ZNSSD criterion. If a POI (shown in red color) was located outside the ROI, as shown in Fig. 7.1, the original POI was artificially moved to a new point (shown in blue color) on the intersecting edge of the ROI.

It is worth noting that, however, the full-field displacement cannot be obtained directly after this procedure, since the POIs were no longer arranged orthogonally in the images. In order to further obtain the full-field displacement, the classical Delaunay triangulation [283] was employed to mesh all the POIs. The full-field deformation was then obtained by scattered data interpolation.

The non-central DIC algorithm enjoys great flexibility and reliability in solving boundary deformation, and it may also be extended to 3D DIC and DVC algorithms

Figure 7.1: Schematic illustration of the non-central DIC method in dealing with the boundary deformation. If the original POI (the red point) falls outside the ROI, it will be moved to a new point (the blue point) on the intersection edge of the ROI.

for broader applications. Nonetheless, it has two inherent problems needing further improvement. First, the continuity of the deformation along the boundary cannot be guaranteed, as the POIs outside the ROI were artificially moved to the edge. Second, the accuracy of the obtained full-field deformation heavily depends on the quality of the triangulation. This issue is even more significant for the 3D DIC and DVC, since the triangulation in 3D space is much more complex than their 2D counterparts.

In fact, a more straightforward way to deal with the boundary deformation is to grab the idea of the finite element method (FEM), as it has been developed to solve the arbitrary shape problems. Finite element (FE) formulation for DIC (FE-DIC) indeed has been developed by Sun et al. in 2005 [56], in which Q4 elements containing four nodes were used in the calculation. However, their Q4 FE-DIC did not intend to handle boundary deformation, and it was designed to compensate the inter-subset discontinuity issue of subset-based DIC algorithms during the correlation process. Subsequently, Ma et al. first applied the idea of the FE-DIC to measure the boundary deformation [58], and they employed Q8 elements which contained 8 nodes per element to improve accuracy and flexibility of the FE-DIC algorithm. The grid defined along the boundary within an image using Q8 elements is schematically shown in Fig. 1.6 where the nodes circulating the

boundary give a good approximation of the boundary shape. Most recently, several further improvements on the performance of the FE-DIC were published. Wang et al. adopted their successful experience in subset-based DIC algorithms into the FE-DIC, which enhanced its accuracy and computation speed. Kleinendorst et al. proposed an adaptive refinement algorithm [284] to refine the mesh used in the FE-DIC, which made the total degrees of freedom in the measurement to be optimized autonomously, ensuring a more accurate results along the boundary area within an image. They argued that it was more reasonable to use more elements along the boundary than other areas within the ROI, and the mesh should be refined where the calculated results were not accurate enough. This type of refinement is very flexible, which makes the method widely applicable to various DIC problems.

As described above, it seems that the FE-DIC should be superior to the subset-based DIC, however, as exhaustively studied by Pan et al. recently, standard subset-based DIC methods outperform FE-DIC methods in terms of both accuracy and computation efficiency when the subset (element) size is not very small [285]. Nonetheless, we found that their comparison between the subset-based DIC and the FE-DIC was incomplete. They only provided the fact that the subset-based DIC outperformed the FE-DIC in processing general images without boundary effects. Thus, the performance difference of measuring boundary deformation between subset-based DIC and FE-DIC approaches remains unclear.

Based on the above discussion, we propose three steps as our future research direction. First, we believe that the FE-DIC is more advanced in handling boundary conditions compared with the subset-based DIC. Thus, we will prove our idea by comparing their performance on images with specific boundaries. Second, we consider that a hybrid FE-DIC and subset-based DIC method should be another good solution in improving robustness. While the FE-DIC may be good at handling boundary deformations, the subset-based DIC can be applied to other areas of an image. The combination not only benefits from the great computation speed of the proposed paDIC method, but also enjoys high accuracy along the boundary.

However, the coupling method between these two areas needs further investigation. Third, as indicated in [285], the computation efficiency of the FE-DIC is much lower than its subset-based counterpart. A GPU-based FE-DIC algorithm has already been attempted by Leclerc et al. [59], however, they used triangles as elements, which is less accurate than Q8 quadrilateral elements in handling complex deformations. We are thus eager to explore the possibility of applying parallel computation to improve the speed of the Q8-element based FE-DIC. We are also considering applying the above three ideas to DVC.

Figure 7.2: Schematic illustration of the hybrid CPU and GPU system framework for fringe pattern analysis.

7.2.3 Accurate and robust 3D real-time DIC system

The proposed paDIC algorithm and rtDIC system can only be applied to 2D in-plane measurement, which is less accurate and robust than 3D DIC, where two cameras are employed to provide a means of measuring the out-of-plane deformation. Therefore, in the near future, we are going to design and develop a 3D real-time DIC system based on rtDIC, which can offer end-users with much more flexibility and freedom in complex 3D measurement yet without losing much speed performance.

7.2.4 Hybrid CPU and GPU applied to fringe pattern analysis

The proposed G-LS3U algorithm has achieved a very high frame rate by using GPU parallel computing. In the future, as shown in Fig. 7.2, to achieve a even high frame rate, we consider to apply the hybrid CPU and GPU parallel computing to further accelerate and optimize G-LS3U as well as other fringe pattern analysis techniques.

References

- [1] A. Shukla and J. W. Dally, *Experimental solid mechanics*. College House Enterprises, LLC, 2014.
- [2] W. D. Callister and D. G. Rethwisch, *Materials science and engineering*, vol. 5. John Wiley & Sons NY, 2011.
- [3] J. Hamill and K. M. Knutzen, *Biomechanical basis of human movement*. Lippincott Williams & Wilkins, 2006.
- [4] D. H. House and J. C. Keyser, *Foundations of Physically Based Modeling and Animation*. CRC Press, 2016.
- [5] M. E. Gurtin, *An introduction to continuum mechanics*, vol. 158. Academic press, 1982.
- [6] T. Chu, W. Ranson, and M. A. Sutton, “Applications of digital-image-correlation techniques to experimental mechanics,” *Experimental mechanics*, vol. 25, no. 3, pp. 232–244, 1985.
- [7] F. Chen, G. M. Brown, and M. Song, “Overview of three-dimensional shape measurement using optical methods,” *Optical Engineering*, vol. 39, no. 1, pp. 10–22, 2000.
- [8] M. A. Otaduy, B. Bickel, D. Bradley, and H. Wang, “Data-driven simulation methods in computer graphics: cloth, tissue and faces,” in *ACM SIGGRAPH 2012 Courses*, p. 12, ACM, 2012.

-
- [9] K. Willam, “Constitutive models for engineering materials,” *Encyclopedia of physical science and technology*, vol. 3, pp. 603–633, 2002.
- [10] M. Müller and M. Gross, “Interactive virtual materials,” in *Proceedings of Graphics Interface 2004*, pp. 239–246, Canadian Human-Computer Communications Society, 2004.
- [11] M. Müller, J. Stam, D. James, and N. Thürey, “Real time physics: class notes,” in *ACM SIGGRAPH 2008 classes*, p. 88, ACM, 2008.
- [12] A. A. Shabana, *Theory of vibration: Volume II: discrete and continuous systems*. Springer Science & Business Media, 2012.
- [13] J. Barbič and D. L. James, “Real-time subspace integration for st. venant-kirchhoff deformable models,” in *ACM transactions on graphics (TOG)*, vol. 24, pp. 982–990, ACM, 2005.
- [14] E. Sifakis and J. Barbic, “Fem simulation of 3d deformable solids: a practitioner’s guide to theory, discretization and model reduction,” in *ACM SIGGRAPH 2012 Courses*, p. 20, ACM, 2012.
- [15] H. Xu, Y. Li, Y. Chen, J. Barbic, *et al.*, “Interactive material design using model reduction,”
- [16] S. Li, J. Huang, F. de Goes, X. Jin, H. Bao, and M. Desbrun, “Space-time editing of elastic motion through material optimization and reduction,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 108, 2014.
- [17] B. Bickel, M. Lang, M. Botsch, M. A. Otaduy, and M. Gross, “Pose-space animation and transfer of facial details,” in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 57–66, Eurographics Association, 2008.

- [18] H. Wu, F. Zhang, S. Cao, S. Xing, and X. Qu, “Absolute distance measurement by intensity detection using a mode-locked femtosecond pulse laser,” *Optics express*, vol. 22, no. 9, pp. 10380–10397, 2014.
- [19] X. Wang, Z. Pan, F. Fan, J. Wang, Y. Liu, S. X. Mao, T. Zhu, and S. Xia, “Nanoscale deformation analysis with high-resolution transmission electron microscopy and digital image correlation,” *Journal of Applied Mechanics*, vol. 82, no. 12, p. 121001, 2015.
- [20] B. Pan, K. Qian, H. Xie, and A. Asundi, “Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review,” *Measurement science and technology*, vol. 20, no. 6, p. 062001, 2009.
- [21] X. Shao, X. Dai, Z. Chen, and X. He, “Real-time 3d digital image correlation method and its application in human pulse monitoring,” *Applied optics*, vol. 55, no. 4, pp. 696–704, 2016.
- [22] W. Peters and W. Ranson, “Digital imaging techniques in experimental stress analysis,” *Optical Engineering*, vol. 21, no. 3, pp. 213427–213427, 1982.
- [23] J. Abanto-Bueno and J. Lambros, “Investigation of crack growth in functionally graded materials using digital image correlation,” *Engineering Fracture Mechanics*, vol. 69, no. 14, pp. 1695–1711, 2002.
- [24] V. Tarigopula, O. S. Hopperstad, M. Langseth, A. H. Clausen, F. Hild, O.-G. Lademo, and M. Eriksson, “A study of large plastic deformations in dual phase steel using digital image correlation and fe analysis,” *Experimental Mechanics*, vol. 48, no. 2, pp. 181–196, 2008.
- [25] V. Savic, L. Hector Jr, and J. Fekete, “Digital image correlation study of plastic deformation and fracture in fully martensitic steels,” *Experimental mechanics*, vol. 50, no. 1, pp. 99–110, 2010.

-
- [26] E. Verhulp, B. van Rietbergen, and R. Huiskes, “A three-dimensional digital image correlation technique for strain measurements in microstructures,” *Journal of biomechanics*, vol. 37, no. 9, pp. 1313–1320, 2004.
- [27] Z. Gao and J. P. Desai, “Estimating zero-strain states of very soft tissue under gravity loading using digital image correlation,” *Medical image analysis*, vol. 14, no. 2, pp. 126–137, 2010.
- [28] L. Horny, H. Chlup, R. Zitny, T. Vonavkova, J. Vesely, and P. Lanzer, “Ex vivo coronary stent implantation evaluated with digital image correlation,” *Experimental mechanics*, vol. 52, no. 9, pp. 1555–1558, 2012.
- [29] L. Chevalier, S. Calloch, F. Hild, and Y. Marco, “Digital image correlation used to analyze the multiaxial behavior of rubber-like materials,” *arXiv preprint physics/0511148*, 2005.
- [30] J. Kang, Y. Ososkov, J. D. Embury, and D. S. Wilkinson, “Digital image correlation studies for microscopic strain distribution and damage in dual phase steels,” *Scripta Materialia*, vol. 56, no. 11, pp. 999–1002, 2007.
- [31] Z.-F. Zhang, Y.-L. Kang, H.-W. Wang, Q.-H. Qin, Y. Qiu, and X.-Q. Li, “A novel coarse-fine search scheme for digital image correlation method,” *Measurement*, vol. 39, no. 8, pp. 710–718, 2006.
- [32] F. Hild, B. Raka, M. Baudequin, S. Roux, and F. Cantelaube, “Multiscale displacement field measurements of compressed mineral-wool samples by digital image correlation,” *Applied optics*, vol. 41, no. 32, pp. 6815–6828, 2002.
- [33] C. Franck, S. Hong, S. Maskarinec, D. Tirrell, and G. Ravichandran, “Three-dimensional full-field measurements of large deformations in soft materials using confocal microscopy and digital volume correlation,” *Experimental Mechanics*, vol. 47, no. 3, pp. 427–438, 2007.

- [34] J. Huang, X. Pan, S. Li, X. Peng, C. Xiong, and J. Fang, “A digital volume correlation technique for 3-d deformation measurements of soft gels,” *International Journal of Applied Mechanics*, vol. 3, no. 02, pp. 335–354, 2011.
- [35] M. Gates, M. T. Heath, and J. Lambros, “High-performance hybrid cpu and gpu parallel algorithm for digital volume correlation,” *International Journal of High Performance Computing Applications*, p. 1094342013518807, 2014.
- [36] M. A. Sutton, J. J. Orteu, and H. Schreier, *Image correlation for shape, motion and deformation measurements: basic concepts, theory and applications*. Springer Science & Business Media, 2009.
- [37] M. A. Sutton, “Computer vision-based, noncontacting deformation measurements in mechanics: a generational transformation,” *Applied Mechanics Reviews*, vol. 65, no. 5, p. 050802, 2013.
- [38] W. Tong, “An evaluation of digital image correlation criteria for strain mapping applications,” *Strain*, vol. 41, no. 4, pp. 167–175, 2005.
- [39] B. Pan, H. Xie, and Z. Wang, “Equivalence of digital image correlation criteria for pattern matching,” *Applied optics*, vol. 49, no. 28, pp. 5501–5509, 2010.
- [40] B. Pan, A. Asundi, H. Xie, and J. Gao, “Digital image correlation using iterative least squares and pointwise least squares for displacement field and strain field measurements,” *Optics and Lasers in Engineering*, vol. 47, no. 7, pp. 865–874, 2009.
- [41] H. Bruck, S. McNeill, M. A. Sutton, and W. Peters Iii, “Digital image correlation using newton-raphson method of partial differential correction,” *Experimental Mechanics*, vol. 29, no. 3, pp. 261–267, 1989.
- [42] P. Bing, X. Hui-Min, X. Bo-Qin, and D. Fu-Long, “Performance of sub-pixel registration algorithms in digital image correlation,” *Measurement Science and Technology*, vol. 17, no. 6, p. 1615, 2006.

-
- [43] G. Vendroux and W. Knauss, "Submicron deformation field measurements: Part 2. improved digital image correlation," *Experimental Mechanics*, vol. 38, no. 2, pp. 86–92, 1998.
- [44] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–1090, IEEE, 2001.
- [45] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.
- [46] B. Pan, K. Li, and W. Tong, "Fast, robust and accurate digital image correlation calculation without redundant computations," *Experimental Mechanics*, vol. 53, no. 7, pp. 1277–1289, 2013.
- [47] B. Pan, "Reliability-guided digital image correlation for image deformation measurement," *Applied optics*, vol. 48, no. 8, pp. 1535–1542, 2009.
- [48] Z. Jiang, Q. Kemao, H. Miao, J. Yang, and L. Tang, "Path-independent digital image correlation with high accuracy, speed and robustness," *Optics and Lasers in Engineering*, vol. 65, pp. 93–102, 2015.
- [49] A. Singh and S. Omkar, "Digital image correlation using gpu computing applied to biomechanics," *Biomed Sci Eng*, vol. 1, pp. 1–10, 2013.
- [50] K. Kapinchev, A. Bradu, F. Barnes, and A. Podoleanu, "Gpu implementation of cross-correlation for image generation in real time," in *Signal Processing and Communication Systems (ICSPCS), 2015 9th International Conference on*, pp. 1–6, IEEE, 2015.
- [51] L. Zhang, T. Wang, Z. Jiang, Q. Kemao, Y. Liu, Z. Liu, L. Tang, and S. Dong, "High accuracy digital image correlation powered by gpu-based

- parallel computing,” *Optics and Lasers in Engineering*, vol. 69, pp. 7–12, 2015.
- [52] G. Le Besnerais, Y. Le Sant, and D. Lvque, “Fast and dense 2d and 3d displacement field estimation by a highly parallel image correlation algorithm,” *Strain*, vol. 52, no. 4, pp. 286–306, 2016.
- [53] B. Pan and K. Li, “A fast digital image correlation method for deformation measurement,” *Optics and Lasers in Engineering*, vol. 49, no. 7, pp. 841–847, 2011.
- [54] Z. Pan, W. Chen, Z. Jiang, L. Tang, Y. Liu, and Z. Liu, “Performance of global look-up table strategy in digital image correlation with cubic b-spline interpolation and bicubic interpolation,” *Theoretical and Applied Mechanics Letters*, vol. 6, no. 3, pp. 126–130, 2016.
- [55] J. Blaber, B. Adair, and A. Antoniou, “Ncorr: open-source 2d digital image correlation matlab software,” *Experimental Mechanics*, vol. 55, no. 6, pp. 1105–1122, 2015.
- [56] Y. Sun, J. H. Pang, C. K. Wong, and F. Su, “Finite element formulation for a digital image correlation method,” *Applied optics*, vol. 44, no. 34, pp. 7357–7363, 2005.
- [57] G. Besnard, F. Hild, and S. Roux, “finite-element displacement fields analysis from digital images: application to portevin–le chatelier bands,” *Experimental Mechanics*, vol. 46, no. 6, pp. 789–803, 2006.
- [58] S. Ma, Z. Zhao, and X. Wang, “Mesh-based digital image correlation method using higher order isoparametric elements,” *The Journal of Strain Analysis for Engineering Design*, vol. 47, no. 3, pp. 163–175, 2012.

-
- [59] H. Leclerc, J.-N. Périé, S. Roux, and F. Hild, “Integrated digital image correlation for the identification of mechanical properties,” in *Computer Vision/Computer Graphics Collaboration Techniques*, pp. 161–171, Springer, 2009.
- [60] L. Wittevrongel, P. Lava, S. V. Lomov, and D. Debruyne, “A self adaptive global digital image correlation algorithm,” *Experimental Mechanics*, vol. 55, no. 2, pp. 361–378, 2015.
- [61] Q. Kemao, “Two-dimensional windowed fourier transform for fringe pattern analysis: principles, applications and implementations,” *Optics and Lasers in Engineering*, vol. 45, no. 2, pp. 304–317, 2007.
- [62] Q. Kemao, *Windowed Fringe Pattern Analysis*. SPIE Press Bellingham, Wash, USA, 2013.
- [63] K. Creath, “Phase-shifting speckle interferometry,” *Applied Optics*, vol. 24, no. 18, pp. 3053–3058, 1985.
- [64] E. P. Goodwin and J. C. Wyant, “Field guide to interferometric optical testing,” SPIE, 2006.
- [65] J. M. Huntley, G. H. Kaufmann, and D. Kerr, “Phase-shifted dynamic speckle pattern interferometry at 1 khz,” *Applied optics*, vol. 38, no. 31, pp. 6556–6563, 1999.
- [66] A. J. van Haasteren and H. J. Frankena, “Real-time displacement measurement using a multicamera phase-stepping speckle interferometer,” *Applied optics*, vol. 33, no. 19, pp. 4137–4142, 1994.
- [67] J. Millerd, N. Brock, J. Hayes, M. North-Morris, B. Kimbrough, and J. Wyant, “Pixelated phase-mask dynamic interferometers,” *Fringe 2005*, pp. 640–647, 2006.

- [68] C. Joenathan, B. Franze, P. Haible, and H. Tiziani, “Speckle interferometry with temporal phase evaluation for measuring large-object deformation,” *Applied optics*, vol. 37, no. 13, pp. 2608–2614, 1998.
- [69] C. Joenathan, B. Franze, P. Haible, and H. Tiziani, “Large in-plane displacement measurement in dual-beam speckle interferometry using temporal phase measurement,” *Journal of modern Optics*, vol. 45, no. 9, pp. 1975–1984, 1998.
- [70] G. H. Kaufmann and G. E. Galizzi, “Phase measurement in temporal speckle pattern interferometry: comparison between the phase-shifting and the fourier transform methods,” *Applied optics*, vol. 41, no. 34, pp. 7254–7263, 2002.
- [71] V. D. Madjarova, H. Kadono, and S. Toyooka, “Dynamic electronic speckle pattern interferometry (despi) phase analyses with temporal hilbert transform,” *Optics express*, vol. 11, no. 6, pp. 617–623, 2003.
- [72] F. A. M. Rodriguez, A. Federico, and G. H. Kaufmann, “Hilbert transform analysis of a time series of speckle interferograms with a temporal carrier,” *Applied optics*, vol. 47, no. 9, pp. 1310–1316, 2008.
- [73] X. Colonna de Lega, “Processing of non-stationary interference patterns: adapted phase shifting algorithms and wavelet analysis,” *Application to dynamic deformation measurements by holographic and speckle interferometry. Theses*, no. 1666, 1997.
- [74] A. Federico and G. H. Kaufmann, “Robust phase recovery in temporal speckle pattern interferometry using a 3d directional wavelet transform,” *Optics letters*, vol. 34, no. 15, pp. 2336–2338, 2009.
- [75] Y. Fu, R. M. Groves, G. Pedrini, and W. Osten, “Kinematic and deformation parameter measurement by spatiotemporal analysis of an interferogram sequence,” *Applied optics*, vol. 46, no. 36, pp. 8645–8655, 2007.

-
- [76] T. E. Carlsson and A. Wei, “Phase evaluation of speckle patterns during continuous deformation by use of phase-shifting speckle interferometry,” *Applied optics*, vol. 39, no. 16, pp. 2628–2637, 2000.
- [77] W. An and T. E. Carlsson, “Speckle interferometry for measurement of continuous deformations,” *Optics and lasers in engineering*, vol. 40, no. 5, pp. 529–541, 2003.
- [78] L. Bruno and A. Poggialini, “Phase shifting speckle interferometry for dynamic phenomena,” *Optics express*, vol. 16, no. 7, pp. 4665–4670, 2008.
- [79] Y. H. Huang, S. P. Ng, L. Liu, Y. S. Chen, and M. Y. Hung, “Shearographic phase retrieval using one single specklegram: a clustering approach,” *Optical Engineering*, vol. 47, no. 5, pp. 054301–054301, 2008.
- [80] Y. Huang, F. Janabi-Sharifi, Y. Liu, and Y. Hung, “Dynamic phase measurement in shearography by clustering method and fourier filtering,” *Optics express*, vol. 19, no. 2, pp. 606–615, 2011.
- [81] L. Kai and Q. Kema, “Dynamic phase retrieval in temporal speckle pattern interferometry using least squares method and windowed fourier filtering,” *Optics express*, vol. 19, no. 19, pp. 18058–18066, 2011.
- [82] Z. Wang and B. Han, “Advanced iterative algorithm for phase extraction of randomly phase-shifted interferograms,” *Optics letters*, vol. 29, no. 14, pp. 1671–1673, 2004.
- [83] T. Wang and K. Qian, “Parallel computing in experimental mechanics and optical measurement: A review (ii),” *Optics and Lasers in Engineering*, 2017.
- [84] T. Wang, K. Qian, H. S. Seah, and F. Lin, “Some thoughts on designing a real-time digital image correlation system, in preparation,” *Applied optics*, 2017.

- [85] T. Wang, L. Kai, and Q. Kemao, “Real-time reference-based dynamic phase retrieval algorithm for optical measurement,” *Appl. Opt.*, vol. 56, pp. 7726–7733, Sep 2017.
- [86] Y. Yu, M. Nakano, and T. Ikeda, “Photomechanics: directed bending of a polymer film by light,” *Nature*, vol. 425, no. 6954, pp. 145–145, 2003.
- [87] L. Kai and Q. Kemao, “Fast frequency-guided sequential demodulation of a single fringe pattern,” *Optics letters*, vol. 35, no. 22, pp. 3718–3720, 2010.
- [88] N. Kehtarnavaz and M. Gamadia, “Real-time image and video processing: from research to reality,” *Synthesis Lectures on Image, Video & Multimedia Processing*, vol. 2, no. 1, pp. 1–108, 2006.
- [89] W. Gao and Q. Kemao, “Parallel computing in experimental mechanics and optical measurement: a review,” *Optics and Lasers in Engineering*, vol. 50, no. 4, pp. 608–617, 2012.
- [90] C. Nvidia, “Programming guide,” 2017.
- [91] A. L. Shimpi *et al.*, “Inside the titan supercomputer: 299k amd x86 cores and 18.6 k nvidia gpus,” *AnandTech online computer hardware magazine*, 2012.
- [92] M. J. Flynn, “Some computer organizations and their effectiveness,” *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.
- [93] R. Duncan, “A survey of parallel computer architectures,” *Computer*, vol. 23, no. 2, pp. 5–16, 1990.
- [94] G. Barlas, *Multicore and GPU Programming: An Integrated Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2015.
- [95] L. Dagum and R. Menon, “Openmp: an industry standard api for shared-memory programming,” *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

-
- [96] J. Reinders, *Intel threading building blocks: outfitting C++ for multi-core processor parallelism.* ” O’Reilly Media, Inc.”, 2007.
- [97] D. Koufaty and D. T. Marr, “Hyperthreading technology in the netburst microarchitecture,” *IEEE Micro*, vol. 23, no. 2, pp. 56–65, 2003.
- [98] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach.* Morgan kaufmann, 2016.
- [99] T. Wang, Z. Jiang, Q. Kemao, F. Lin, and S. Soon, “Gpu accelerated digital volume correlation,” *Experimental Mechanics*, vol. 56, no. 2, pp. 297–309, 2016.
- [100] J. W. Cooley, P. A. Lewis, and P. D. Welch, “The fast fourier transform and its applications,” *IEEE Transactions on Education*, vol. 12, no. 1, pp. 27–34, 1969.
- [101] H. Mark, “Optimizing parallel reduction in cuda,” *NVIDIA CUDA SDK*, vol. 2, 2008.
- [102] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3.* Addison-Wesley, 2013.
- [103] D. Ruijters, B. M. ter Haar Romeny, and P. Suetens, “Efficient gpu-based texture interpolation using uniform b-splines,” *Journal of Graphics Tools*, vol. 13, no. 4, pp. 61–69, 2008.
- [104] D. Ruijters and P. Thévenaz, “Gpu prefilter for accurate cubic b-spline interpolation,” *The Computer Journal*, vol. 55, no. 1, pp. 15–20, 2010.
- [105] F. Champagnat and Y. Le Sant, “Efficient cubic b-spline image interpolation on a gpu,” *Journal of Graphics Tools*, vol. 16, no. 4, pp. 218–232, 2012.

- [106] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, “Intel math kernel library,” in *High-Performance Computing on the Intel® Xeon Phi*, pp. 167–188, Springer, 2014.
- [107] M. Frigo and S. G. Johnson, “Fftw: An adaptive software architecture for the fft,” in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3, pp. 1381–1384, IEEE, 1998.
- [108] C. Nvidia, “Cublas library,” *NVIDIA Corporation, Santa Clara, California*, vol. 15, no. 27, p. 31, 2008.
- [109] C. Nvidia, “Cusolver library,” 2017.
- [110] C. Nvidia, “Cufft library,” 2010.
- [111] N. Bell and J. Hoberock, “Thrust: A productivity-oriented library for cuda,” *GPU computing gems Jade edition*, vol. 2, pp. 359–371, 2011.
- [112] J. E. Stone, D. Gohara, and G. Shi, “Opencl: A parallel programming standard for heterogeneous computing systems,” *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.
- [113] B. K. Bay, T. S. Smith, D. P. Fyhrie, and M. Saad, “Digital volume correlation: three-dimensional strain mapping using x-ray tomography,” *Experimental mechanics*, vol. 39, no. 3, pp. 217–226, 1999.
- [114] Z. Wang, S. Wang, and Z. Wang, “An analysis on computational load of dic based on newton–raphson scheme,” *Optics and Lasers in Engineering*, vol. 52, pp. 61–65, 2014.
- [115] D. Gembris, M. Neeb, M. Gipp, A. Kugel, and R. Männer, “Correlation analysis on gpu systems using nvidias cuda,” *Journal of real-time image processing*, vol. 6, no. 4, pp. 275–280, 2011.

-
- [116] B. Marciniak, T. Marciniak, Z. Lutowski, and S. Bujnowski, “Usage of digital image correlation in analysis of cracking processes,” *Image Processing & Communications*, vol. 17, no. 3, pp. 21–28, 2012.
- [117] X. Shao, X. Dai, and X. He, “Noise robustness and parallel computation of the inverse compositional gauss–newton algorithm in digital image correlation,” *Optics and Lasers in Engineering*, vol. 71, pp. 9–19, 2015.
- [118] R. Wu, C. Kong, K. Li, and D. Zhang, “Real-time digital image correlation for dynamic strain measurement,” *Experimental Mechanics*, vol. 56, no. 5, pp. 833–843, 2016.
- [119] B. Pan and L. Tian, “Superfast robust digital image correlation analysis with parallel computing,” *Optical Engineering*, vol. 54, no. 3, pp. 034106–034106, 2015.
- [120] S. Baker, R. Patil, K. M. Cheung, and I. Matthews, “Lucas-kanade 20 years on: Part 5,” *Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-04-64*, 2004.
- [121] I. Jandejsek, O. Jiroušek, and D. Vavřík, “Precise strain measurement in complex materials using digital volumetric correlation and time lapse micro-ct data,” *Procedia Engineering*, vol. 10, pp. 1730–1735, 2011.
- [122] V. Valle, S. Hedan, P. Cosenza, A. Fauchille, and M. Berdjane, “Digital image correlation development for the study of materials including multiple crossing cracks,” *Experimental Mechanics*, vol. 55, no. 2, pp. 379–391, 2015.
- [123] E. Bar-Kochba, J. Toyjanova, E. Andrews, K.-S. Kim, and C. Franck, “A fast iterative digital volume correlation algorithm for large deformations,” *Experimental Mechanics*, vol. 55, no. 1, pp. 261–274, 2015.
- [124] E. Bar-Kochba, J. Toyjanova, E. Andrews, K.-S. Kim, and C. Franck, “A fast iterative digital volume correlation algorithm for large deformations,” *Experimental Mechanics*, vol. 55, no. 1, pp. 261–274, 2014.

- [125] M. Gates, J. Lambros, and M. Heath, “Towards high performance digital volume correlation,” *Experimental Mechanics*, vol. 51, no. 4, pp. 491–507, 2011.
- [126] M. Gates, M. T. Heath, and J. Lambros, “High-performance hybrid cpu and gpu parallel algorithm for digital volume correlation,” *The International Journal of High Performance Computing Applications*, vol. 29, no. 1, pp. 92–106, 2015.
- [127] F. Champagnat, A. Plyer, G. Le Besnerais, B. Leclaire, S. Davoust, and Y. Le Sant, “Fast and accurate piv computation using highly parallel iterative correlation maximization,” *Experiments in fluids*, vol. 50, no. 4, p. 1169, 2011.
- [128] H. Leclerc, J.-N. Périé, F. Hild, and S. Roux, “Digital volume correlation: what are the limits to the spatial resolution?,” *Mechanics & Industry*, vol. 13, no. 6, pp. 361–371, 2012.
- [129] H. Leclerc, S. Roux, and F. Hild, “Projection savings in ct-based digital volume correlation,” *Experimental Mechanics*, vol. 55, no. 1, pp. 275–287, 2015.
- [130] J.-C. Passieux, J.-N. Périé, and M. Salaün, “A dual domain decomposition method for finite element digital image correlation,” *International Journal for Numerical Methods in Engineering*, vol. 102, no. 10, pp. 1670–1682, 2015.
- [131] R. Bouclier and J.-C. Passieux, “A domain coupling method for finite element digital image correlation with mechanical regularization: Application to multiscale measurements and parallel computing,” *International Journal for Numerical Methods in Engineering*, vol. 111, no. 2, pp. 123–143, 2017.
- [132] N. Karpinsky and S. Zhang, “High-resolution, real-time 3d imaging with fringe analysis,” *Journal of Real-Time Image Processing*, vol. 7, no. 1, pp. 55–66, 2012.

-
- [133] N. Karpinsky and S. Zhang, “3d range geometry video compression with the h. 264 codec,” *Optics and Lasers in Engineering*, vol. 51, no. 5, pp. 620–625, 2013.
- [134] N. Karpinsky, M. Hoke, V. Chen, and S. Zhang, “High-resolution, real-time three-dimensional shape measurement on graphics processing unit,” *Optical Engineering*, vol. 53, no. 2, pp. 024105–024105, 2014.
- [135] H. Nguyen, D. Nguyen, Z. Wang, H. Kieu, and M. Le, “Real-time, high-accuracy 3d imaging and shape measurement,” *Applied optics*, vol. 54, no. 1, pp. A9–A17, 2015.
- [136] V. Demir and A. Z. Elsherbeni, “Utilization of cuda-opengl interoperability to display electromagnetic fields calculated by fdtd,” in *Computational Electromagnetics International Workshop (CEM)*, 2011, pp. 95–98, IEEE, 2011.
- [137] S. Van der Jeught, J. A. Soons, and J. J. Dirckx, “Real-time microscopic phase-shifting profilometry,” *Applied optics*, vol. 54, no. 15, pp. 4953–4959, 2015.
- [138] S. Van der Jeught and J. J. Dirckx, “Real-time structured light profilometry: a review,” *Optics and lasers in engineering.-Barking*, pp. 1–31, 2016.
- [139] M. Zhao and Q. Kemao, “Multicore implementation of the windowed fourier transform algorithms for fringe pattern analysis,” *Applied Optics*, vol. 54, no. 3, pp. 587–594, 2015.
- [140] H. Zhong, J. Tang, and S. Zhang, “Phase quality map based on local multi-unwrapped results for two-dimensional phase unwrapping,” *Applied optics*, vol. 54, no. 4, pp. 739–745, 2015.
- [141] A. Zimmer and P. Ghuman, “Cuda optimization of non-local means extended to wrapped gaussian distributions for interferometric phase denoising,” *Procedia Computer Science*, vol. 80, pp. 166–177, 2016.

- [142] X. Zhu, Y. Wu, and F. Liu, “Gpu-accelerated phase extraction algorithm for interferograms: a real-time application,” in *SPIE/COS Photonics Asia*, pp. 1002327–1002327, International Society for Optics and Photonics, 2016.
- [143] O. Backoach, S. Kariv, P. Girshovitz, and N. T. Shaked, “Fast phase processing in off-axis holography by cuda including parallel phase unwrapping,” *Optics express*, vol. 24, no. 4, pp. 3177–3188, 2016.
- [144] G. Dardikman, M. Habaza, L. Waller, and N. T. Shaked, “Video-rate processing in tomographic phase microscopy of biological cells using cuda,” *Optics express*, vol. 24, no. 11, pp. 11839–11854, 2016.
- [145] J. Li, P. Bloch, J. Xu, M. V. Sarunic, and L. Shannon, “Performance and scalability of fourier domain optical coherence tomography acceleration using graphics processing units,” *Applied optics*, vol. 50, no. 13, pp. 1832–1838, 2011.
- [146] Y. Jian, K. Wong, and M. V. Sarunic, “Graphics processing unit accelerated optical coherence tomography processing at megahertz axial scan rate and high resolution video rate volumetric rendering,” *Journal of biomedical optics*, vol. 18, no. 2, pp. 026002–026002, 2013.
- [147] Y. Jian, J. Xu, M. A. Gradowski, S. Bonora, R. J. Zawadzki, and M. V. Sarunic, “Wavefront sensorless adaptive optics optical coherence tomography for in vivo retinal imaging in mice,” *Biomedical optics express*, vol. 5, no. 2, pp. 547–559, 2014.
- [148] K. S. Wong, Y. Jian, M. Cua, S. Bonora, R. J. Zawadzki, and M. V. Sarunic, “In vivo imaging of human photoreceptor mosaic with wavefront sensorless adaptive optics optical coherence tomography,” *Biomedical optics express*, vol. 6, no. 2, pp. 580–590, 2015.

-
- [149] Y. Wang, C. M. Oh, M. C. Oliveira, M. S. Islam, A. Ortega, and B. H. Park, "Gpu accelerated real-time multi-functional spectral-domain optical coherence tomography system at 1300nm," *Optics express*, vol. 20, no. 14, pp. 14797–14813, 2012.
- [150] Y. Huang, X. Liu, and J. U. Kang, "Real-time 3d and 4d fourier domain doppler optical coherence tomography based on dual graphics processing units," *Biomedical optics express*, vol. 3, no. 9, pp. 2162–2174, 2012.
- [151] Y. Huang, X. Liu, C. Song, and J. U. Kang, "Motion-compensated hand-held common-path fourier-domain optical coherence tomography probe for image-guided intervention," *Biomedical optics express*, vol. 3, no. 12, pp. 3105–3118, 2012.
- [152] L. Wang, B. Hofer, J. A. Guggenheim, and B. Považay, "Graphics processing unit-based dispersion encoded full-range frequency-domain optical coherence tomography," *Journal of biomedical optics*, vol. 17, no. 7, pp. 077007–077007, 2012.
- [153] N. H. Cho, U. Jung, S. Kim, W. Jung, J. Oh, H. W. Kang, and J. Kim, "High speed sd-oct system using gpu accelerated mode for in vivo human eye imaging," *Journal of the Optical Society of Korea*, vol. 17, no. 1, pp. 68–72, 2013.
- [154] N. H. Cho, K. Park, J.-Y. Kim, Y. Jung, and J. Kim, "Quantitative assessment of touch-screen panel by nondestructive inspection with three-dimensional real-time display optical coherence tomography," *Optics and Lasers in Engineering*, vol. 68, pp. 50–57, 2015.
- [155] S. J. Wright, R. D. Nowak, and M. A. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2479–2493, 2009.

- [156] D. Xu, Y. Huang, and J. U. Kang, “Real-time compressive sensing spectral domain optical coherence tomography,” *Optics letters*, vol. 39, no. 1, pp. 76–79, 2014.
- [157] D. Xu, Y. Huang, and J. U. Kang, “Gpu-accelerated non-uniform fast fourier transform-based compressive sensing spectral domain optical coherence tomography,” *Optics express*, vol. 22, no. 12, pp. 14871–14884, 2014.
- [158] D. Xu, Y. Huang, and J. U. Kang, “Real-time dispersion-compensated image reconstruction for compressive sensing spectral domain optical coherence tomography,” *JOSA A*, vol. 31, no. 9, pp. 2064–2069, 2014.
- [159] Y. Wang, Y. Wang, A. Akansu, K. D. Belfield, B. Hubbi, and X. Liu, “Robust motion tracking based on adaptive speckle decorrelation analysis of oct signal,” *Biomedical optics express*, vol. 6, no. 11, pp. 4302–4316, 2015.
- [160] M. Zhang, L. Ma, and P. Yu, “Spatial convolution for mirror image suppression in fourier domain optical coherence tomography,” *Optics Letters*, vol. 42, no. 3, pp. 506–509, 2017.
- [161] L. Lim, G. Cheung, and S. Lee, “Comparison of spectral domain and swept-source optical coherence tomography in pathological myopia,” *Eye*, vol. 28, no. 4, p. 488, 2014.
- [162] K. K. Lee, A. Mariampillai, X. Joe, D. W. Cadotte, B. C. Wilson, B. A. Standish, and V. X. Yang, “Real-time speckle variance swept-source optical coherence tomography using a graphics processing unit,” *Biomedical optics express*, vol. 3, no. 7, pp. 1557–1564, 2012.
- [163] D. MacDougall, J. Farrell, J. Brown, M. Bance, and R. Adamson, “Long-range, wide-field swept-source optical coherence tomography with gpu accelerated digital lock-in doppler vibrography for real-time, in vivo middle ear diagnostics,” *Biomedical optics express*, vol. 7, no. 11, pp. 4621–4635, 2016.

-
- [164] C. Viehland, B. Keller, O. M. Carrasco-Zevallos, D. Nankivil, L. Shen, S. Mangalesh, A. N. Kuo, C. A. Toth, J. A. Izatt, *et al.*, “Enhanced volumetric visualization for real time 4d intraoperative ophthalmic swept-source oct,” *Biomedical optics express*, vol. 7, no. 5, pp. 1815–1829, 2016.
- [165] A. G. Podoleanu and A. Bradu, “Master–slave interferometry for parallel spectral domain interferometry sensing and versatile 3d optical coherence tomography,” *Optics express*, vol. 21, no. 16, pp. 19324–19338, 2013.
- [166] A. Bradu and A. G. Podoleanu, “Imaging the eye fundus with real-time en-face spectral domain optical coherence tomography,” *Biomedical optics express*, vol. 5, no. 4, pp. 1233–1249, 2014.
- [167] A. Bradu, K. Kapinchev, F. Barnes, and A. Podoleanu, “Master slave en-face oct/slo,” *Biomedical optics express*, vol. 6, no. 9, pp. 3655–3669, 2015.
- [168] R. Cernat, A. Bradu, N. M. Israelsen, O. Bang, S. Rivet, P. A. Keane, D.-G. Heath, R. Rajendram, and A. Podoleanu, “Gabor fusion master slave optical coherence tomography,” *Biomedical optics express*, vol. 8, no. 2, pp. 813–827, 2017.
- [169] T. Zhang, “Accelerating finite-element-based 3d optical imaging reconstruction by graphic processing units,” in *Biomedical Optics*, pp. BTu3A–55, Optical Society of America, 2012.
- [170] X. Yi, X. Wang, W. Chen, W. Wan, H. Zhao, and F. Gao, “Full domain-decomposition scheme for diffuse optical tomography of large-sized tissues with a combined cpu and gpu parallelization,” *Applied optics*, vol. 53, no. 13, pp. 2754–2765, 2014.
- [171] X. Jiang, Y. Deng, Z. Luo, and Q. Luo, “Accelerating fdot image reconstruction based on path-history fluorescence monte carlo model by using three-level parallel architecture,” *Optics express*, vol. 23, no. 20, pp. 25996–26011, 2015.

- [172] Y. Sung, W. Choi, C. Fang-Yen, K. Badizadegan, R. R. Dasari, and M. S. Feld, “Optical diffraction tomography for high resolution live cell imaging,” *Optics express*, vol. 17, no. 1, pp. 266–277, 2009.
- [173] K. Kim, K. S. Kim, H. Park, J. C. Ye, and Y. Park, “Real-time visualization of 3-d dynamic microscopic objects using optical diffraction tomography,” *Optics express*, vol. 21, no. 26, pp. 32269–32278, 2013.
- [174] W. Gao, Q. Kemao, H. Wang, F. Lin, and H. S. Seah, “Parallel computing for fringe pattern processing: A multicore cpu approach in matlab® environment,” *Optics and Lasers in Engineering*, vol. 47, no. 11, pp. 1286–1292, 2009.
- [175] J. Lobera, G. Ortega, I. García, M. del Pilar Arroyo, and E. M. Garzón, “High performance computing for a 3-d optical diffraction tomographic application in fluid velocimetry,” *Optics express*, vol. 23, no. 4, pp. 4021–4032, 2015.
- [176] M. Xu and L. V. Wang, “Photoacoustic imaging in biomedicine,” *Review of scientific instruments*, vol. 77, no. 4, p. 041101, 2006.
- [177] A. Ozbek, X. Deán-Ben, and D. Razansky, “Realtime parallel back-projection algorithm for three-dimensional optoacoustic imaging devices,” in *European Conference on Biomedical Optics*, p. 88000I, Optical Society of America, 2013.
- [178] K. Peng, L. He, Z. Zhu, J. Tang, and J. Xiao, “Three-dimensional photoacoustic tomography based on graphics-processing-unit-accelerated finite element method,” *Applied optics*, vol. 52, no. 34, pp. 8270–8279, 2013.
- [179] H. Kang, S.-W. Lee, E.-S. Lee, S.-H. Kim, and T. G. Lee, “Real-time gpu-accelerated processing and volumetric display for wide-field laser-scanning optical-resolution photoacoustic microscopy,” *Biomedical optics express*, vol. 6, no. 12, pp. 4650–4660, 2015.

-
- [180] Wikipedia, “Hyperspectral imaging — wikipedia, the free encyclopedia,” 2017. [Online; accessed 24-September-2017].
- [181] J. Solomon and B. Rock, “Imaging spectrometry for earth remote sensing,” *Science*, vol. 228, no. 4704, pp. 1147–1152, 1985.
- [182] M. Ciznicki, K. Kurowski, and A. Plaza, “Graphics processing unit implementation of jpeg2000 for hyperspectral image compression,” *Journal of Applied Remote Sensing*, vol. 6, no. 1, pp. 061507–1, 2012.
- [183] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, “Real-time implementation of remotely sensed hyperspectral image unmixing on gpus,” *Journal of Real-Time Image Processing*, vol. 10, no. 3, pp. 469–483, 2015.
- [184] Y. Wu, L. Gao, B. Zhang, H. Zhao, and J. Li, “Real-time implementation of optimized maximum noise fraction transform for feature extraction of hyperspectral images,” *Journal of Applied Remote Sensing*, vol. 8, no. 1, pp. 084797–084797, 2014.
- [185] Y.-L. Chang, J.-N. Liu, Y.-L. Chen, W.-Y. Chang, T.-J. Hsieh, and B. Huang, “Hyperspectral band selection based on parallel particle swarm optimization and impurity function band prioritization schemes,” *Journal of Applied Remote Sensing*, vol. 8, no. 1, pp. 084798–084798, 2014.
- [186] S. Bernabé, G. Martín, J. M. Nascimento, J. M. Bioucas-Dias, A. Plaza, and V. Silva, “Parallel hyperspectral coded aperture for compressive sensing on gpus,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 932–944, 2016.
- [187] S. Bernabé, G. Martín, J. M. Nascimento, J. M. Bioucas-Dias, A. Plaza, and V. Silva, “Gpu implementation of a hyperspectral coded aperture algorithm for compressive sensing,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, pp. 521–524, IEEE, 2015.

- [188] J. Sevilla, G. Martín, J. Nascimento, and J. Bioucas-Dias, “Hyperspectral image reconstruction from random projections on gpu,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*, pp. 280–283, IEEE, 2016.
- [189] S. Bernabe, S. Sanchez, A. Plaza, S. López, J. A. Benediktsson, and R. Sarmiento, “Hyperspectral unmixing on gpus and multi-core processors: A comparison,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 3, pp. 1386–1398, 2013.
- [190] A. Remón, S. Sánchez, S. Bernabé, E. S. Quintana-Ortí, and A. Plaza, “Performance versus energy consumption of hyperspectral unmixing algorithms on multi-core platforms,” *EURASIP Journal on Advances in Signal Processing*, vol. 2013, no. 1, p. 68, 2013.
- [191] S. Sánchez and A. Plaza, “Fast determination of the number of endmembers for real-time hyperspectral unmixing on gpus,” *Journal of real-time image processing*, vol. 9, no. 3, pp. 397–405, 2014.
- [192] E. Torti, M. Acquistapace, G. Danese, F. Leporati, and A. Plaza, “Real-time identification of hyperspectral subspaces,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2680–2687, 2014.
- [193] P. Benner, V. Novaković, A. Plaza, E. S. Quintana-Ortí, and A. Remón, “Fast and reliable noise estimation for hyperspectral subspace identification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 6, pp. 1199–1203, 2015.
- [194] E. Torti, G. Danese, F. Leporati, and A. Plaza, “A hybrid cpu–gpu real-time hyperspectral unmixing chain,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 945–951, 2016.

-
- [195] X. Wu, B. Huang, L. Wang, and J. Zhang, “Gpu-based parallel design of the hyperspectral signal subspace identification by minimum error (hysime),” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 9, pp. 4400–4406, 2016.
- [196] S. Bernabé, A. Plaza, S. López, and R. Sarmiento, “Parallel implementation of a hyperspectral unmixing chain: Graphic processing units versus multi-core processors,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pp. 3463–3466, IEEE, 2012.
- [197] J. Sevilla, S. Bernabe, and A. Plaza, “Unmixing-based content retrieval system for remotely sensed hyperspectral imagery on gpus,” *The Journal of Supercomputing*, vol. 70, no. 2, pp. 588–599, 2014.
- [198] Z. Wu, S. Ye, J. Wei, Z. Wei, L. Sun, and J. Liu, “Fast endmember extraction for massive hyperspectral sensor data on gpus,” *International Journal of Distributed Sensor Networks*, vol. 9, no. 10, p. 217180, 2013.
- [199] S. Sánchez and A. Plaza, “Parallel hyperspectral image compression using iterative error analysis on graphics processing units,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pp. 3474–3477, IEEE, 2012.
- [200] C. Gonzalez, S. Sánchez, A. Paz, J. Resano, D. Mozos, and A. Plaza, “Use of fpga or gpu-based architectures for remotely sensed hyperspectral image processing,” *INTEGRATION, the VLSI journal*, vol. 46, no. 2, pp. 89–103, 2013.
- [201] E. M. Sigurdsson, A. Plaza, and J. A. Benediktsson, “Gpu implementation of iterative-constrained endmember extraction from remotely sensed hyperspectral images,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2939–2949, 2015.

- [202] A. Agathos, J. Li, D. Petcu, and A. Plaza, “Multi-gpu implementation of the minimum volume simplex analysis algorithm for hyperspectral unmixing,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2281–2296, 2014.
- [203] G. M. Callicó, S. Lopez, B. Aguilar, J. F. López, and R. Sarmiento, “Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using opencl,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 8, pp. 3650–3659, 2014.
- [204] J. Sevilla and J. M. Nascimento, “Parallel gpu architecture for hyperspectral unmixing based on augmented lagrangian method,” in *EUROCON 2015-International Conference on Computer as a Tool (EUROCON)*, IEEE, pp. 1–6, IEEE, 2015.
- [205] J. Delgado, G. Martin, J. Plaza, L. I. Jimenez, and A. Plaza, “Gpu implementation of spatial preprocessing for spectral unmixing of hyperspectral data,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, pp. 5043–5046, IEEE, 2015.
- [206] J. Delgado, G. Martín, J. Plaza, L.-I. Jiménez, and A. Plaza, “On the optimization of memory access to increase the performance of spatial preprocessing techniques on graphics processing units,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*, pp. 6541–6544, IEEE, 2016.
- [207] L.-I. Jiménez, G. Martín, S. Sánchez, J. Plaza, and A. Plaza, “Fast spatial-spectral preprocessing for endmember extraction and spectral unmixing using graphic processing units,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*, pp. 7038–7041, IEEE, 2016.

-
- [208] E. Martel, R. Guerra, S. López, and R. Sarmiento, “A gpu-based processing chain for linearly unmixing hyperspectral images,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 3, pp. 818–834, 2017.
- [209] I. S. Reed and X. Yu, “Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 10, pp. 1760–1770, 1990.
- [210] J. M. Molero, E. M. Garzón, I. García, and A. Plaza, “Analysis and optimizations of global and local versions of the rx algorithm for anomaly detection in hyperspectral data,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 2, pp. 801–814, 2013.
- [211] J. M. Molero, E. M. Garzón, I. García, and A. Plaza, “Anomaly detection based on a parallel kernel rx algorithm for multicore platforms,” *Journal of Applied Remote Sensing*, vol. 6, no. 1, pp. 061503–1, 2012.
- [212] J. M. Molero, E. M. Garzón, I. García, E. S. Quintana-Ortí, and A. Plaza, “Efficient implementation of hyperspectral anomaly detection techniques on gpus and multicore processors,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2256–2266, 2014.
- [213] C. Zhao, W. You, Y. Wang, and J. Wang, “Gpu implementation for real-time hyperspectral anomaly detection,” in *Digital Signal Processing (DSP), 2015 IEEE International Conference on*, pp. 940–943, IEEE, 2015.
- [214] S. Bernabe, S. López, A. Plaza, and R. Sarmiento, “Gpu implementation of an automatic target detection and classification algorithm for hyperspectral image analysis,” *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 2, pp. 221–225, 2013.

- [215] J. López-Fandiño, P. Quesada-Barriuso, D. B. Heras, and F. Argüello, “Efficient elm-based techniques for the classification of hyperspectral remote sensing images on commodity gpus,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2884–2893, 2015.
- [216] Z. Wu, J. Liu, A. Plaza, J. Li, and Z. Wei, “Gpu implementation of composite kernels for hyperspectral image classification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 9, pp. 1973–1977, 2015.
- [217] K. Tan, J. Zhang, Q. Du, and X. Wang, “Gpu parallel implementation of support vector machines for hyperspectral image classification,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4647–4656, 2015.
- [218] P. Quesada-Barriuso, F. Argüello, D. B. Heras, and J. A. Benediktsson, “Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2962–2970, 2015.
- [219] Y. Wu, J. Li, L. Gao, X. Tan, and B. Zhang, “Graphics processing unit-accelerated computation of the markov random fields and loopy belief propagation algorithms for hyperspectral image classification,” *Journal of Applied Remote Sensing*, vol. 9, no. 1, pp. 097295–097295, 2015.
- [220] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, and Z. Wei, “Real-time implementation of the sparse multinomial logistic regression for hyperspectral image classification on gpus,” *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 7, pp. 1456–1460, 2015.
- [221] S. Lopez, T. Vladimirova, C. Gonzalez, J. Resano, D. Mozos, and A. Plaza, “The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends,” *Proceedings of the IEEE*, vol. 101, no. 3, pp. 698–722, 2013.

-
- [222] M. S. Brown, E. Glaser, S. Grassinger, A. Slone, and M. Salvador, “Development of an efficient automated hyperspectral processing system using embedded computing,” in *SPIE Defense, Security, and Sensing*, pp. 839018–839018, International Society for Optics and Photonics, 2012.
- [223] Y. Wu, B. Zhang, H. Zhao, J. Gao, L. Ni, W. Yang, *et al.*, “Gpus for parallel on-board hyperspectral image radiometric normalization,” in *Proc. of SPIE Vol.*, vol. 8743, pp. 874322–1, 2013.
- [224] Y. Wu, L. Gao, B. Zhang, B. Yang, and Z. Chen, “Embedded gpu implementation of anomaly detection for hyperspectral images,” in *SPIE Remote Sensing*, pp. 964608–964608, International Society for Optics and Photonics, 2015.
- [225] L. Zhang, J. B. Gao, Y. Hu, K. F. Sun, Y. H. Wang, J. Cheng, D. D. Sun, and Y. Li, “A new embedded solution of hyper-spectral data processing platform: the embedded gpu computer,” in *International Symposium on Optoelectronic Technology and Application 2016*, pp. 101560M–101560M, International Society for Optics and Photonics, 2016.
- [226] J. Weng, T. Shimobaba, N. Okada, H. Nakayama, M. Oikawa, N. Masuda, and T. Ito, “Generation of real-time large computer generated hologram using wavefront recording method,” *Optics Express*, vol. 20, no. 4, pp. 4018–4023, 2012.
- [227] A.-H. Phan, M.-l. Piao, S.-K. Gil, and N. Kim, “Generation speed and reconstructed image quality enhancement of a long-depth object using double wavefront recording planes and a gpu,” *Applied optics*, vol. 53, no. 22, pp. 4817–4824, 2014.
- [228] F. Yang, A. Kaczorowski, and T. D. Wilkinson, “Fast precalculated triangular mesh algorithm for 3d binary computer-generated holograms,” *Applied optics*, vol. 53, no. 35, pp. 8261–8267, 2014.

- [229] R. Watanabe, K. Yamaguchi, and Y. Sakamoto, “Fast calculation method of computer generated hologram animation for viewpoint parallel shift and rotation using fourier transform optical system,” *Applied optics*, vol. 55, no. 3, pp. A167–A177, 2016.
- [230] G. Makey, M. S. El-Daher, and K. Al-Shufi, “Modification of common fourier computer generated hologram’s representation methods from sequential to parallel computing,” *Optik-International Journal for Light and Electron Optics*, vol. 126, no. 11, pp. 1067–1071, 2015.
- [231] Y.-H. Lee, Y.-H. Seo, J.-S. Yoo, and D.-W. Kim, “High-performance computer-generated hologram by optimized implementation of parallel gpgpus,” *Journal of the Optical Society of Korea*, vol. 18, no. 6, pp. 698–705, 2014.
- [232] Y. Zhang, J. Liu, X. Li, and Y. Wang, “Fast processing method to generate gigabyte computer generated holography for three-dimensional dynamic holographic display,” *Chinese Optics Letters*, vol. 14, no. 3, p. 030901, 2016.
- [233] H. Wei, G. Gong, and N. Li, “Improved look-up table method of computer-generated holograms,” *Applied optics*, vol. 55, no. 32, pp. 9255–9264, 2016.
- [234] M.-W. Kwon, S.-C. Kim, S.-E. Yoon, Y.-S. Ho, and E.-S. Kim, “Object tracking mask-based nlut on gpus for real-time generation of holographic videos of three-dimensional scenes,” *Optics express*, vol. 23, no. 3, pp. 2101–2120, 2015.
- [235] Y. Ogihara and Y. Sakamoto, “Fast calculation method of a cgh for a patch model using a point-based method,” *Applied optics*, vol. 54, no. 1, pp. A76–A83, 2015.
- [236] T. Sugawara, Y. Ogihara, and Y. Sakamoto, “Fast point-based method of a computer-generated hologram for a triangle-patch model by using a graphics processing unit,” *Applied optics*, vol. 55, no. 3, pp. A160–A166, 2016.

-
- [237] A. Gilles, P. Gioia, R. Cozot, and L. Morin, “Hybrid approach for fast occlusion processing in computer-generated hologram calculation,” *Applied optics*, vol. 55, no. 20, pp. 5459–5470, 2016.
- [238] M. Plöschner, B. Straka, K. Dholakia, and T. Čižmár, “Gpu accelerated toolbox for real-time beam-shaping in multimode fibres,” *Optics express*, vol. 22, no. 3, pp. 2933–2947, 2014.
- [239] G. Vizsnyiczai, L. Kelemen, and P. Ormos, “Holographic multi-focus 3d two-photon polymerization with real-time calculated holograms,” *Optics express*, vol. 22, no. 20, pp. 24217–24223, 2014.
- [240] M. Plöschner and T. Čižmár, “Compact multimode fiber beam-shaping system based on gpu accelerated digital holography,” *Optics letters*, vol. 40, no. 2, pp. 197–200, 2015.
- [241] Y. K. Lam, W. Situ, and P. Tsang, “Fast compression of computer-generated holographic images based on a gpu-accelerated skip-dimension vector quantization method,” *Chinese Optics Letters*, vol. 11, no. 5, p. 050901, 2013.
- [242] Y. Endo, T. Shimobaba, T. Kakue, and T. Ito, “Gpu-accelerated compressive holography,” *Optics express*, vol. 24, no. 8, pp. 8437–8445, 2016.
- [243] K.-M. Jeong, H.-S. Kim, S.-I. Hong, S.-K. Lee, N.-Y. Jo, Y.-S. Kim, H.-G. Lim, and J.-H. Park, “Acceleration of integral imaging based incoherent fourier hologram capture using graphic processing unit,” *Optics express*, vol. 20, no. 21, pp. 23735–23743, 2012.
- [244] F. Yi, I. Moon, J.-A. Lee, and B. Javidi, “Fast 3d computational integral imaging using graphics processing unit,” *Journal of Display Technology*, vol. 8, no. 12, pp. 714–722, 2012.
- [245] F. Yi, J. Lee, and I. Moon, “Simultaneous reconstruction of multiple depth images without off-focus points in integral imaging using a graphics processing unit,” *Applied optics*, vol. 53, no. 13, pp. 2777–2786, 2014.

- [246] K.-C. Kwon, C. Park, M.-U. Erdenebat, J.-S. Jeong, J.-H. Choi, N. Kim, J.-H. Park, Y.-T. Lim, and K.-H. Yoo, “High speed image space parallel processing for computer-generated integral imaging system,” *Optics express*, vol. 20, no. 2, pp. 732–740, 2012.
- [247] K.-C. Kwon, J.-S. Jeong, M.-U. Erdenebat, Y.-T. Lim, K.-H. Yoo, and N. Kim, “Real-time interactive display for integral imaging microscopy,” *Applied optics*, vol. 53, no. 20, pp. 4450–4459, 2014.
- [248] K.-C. Kwon, J.-S. Jeong, M.-U. Erdenebat, Y.-L. Piao, K.-H. Yoo, and N. Kim, “Resolution-enhancement for an orthographic-view image display in an integral imaging microscope system,” *Biomedical optics express*, vol. 6, no. 3, pp. 736–746, 2015.
- [249] S. for Experimental Mechanics, “Digital image correlation challenge, sample12,” 2002.
- [250] T. S. Smith, B. K. Bay, and M. M. Rashid, “Digital volume correlation including rotational degrees of freedom during minimization,” *Experimental Mechanics*, vol. 42, no. 3, pp. 272–278, 2002.
- [251] R. Zauel, Y. Yeni, B. Bay, X. Dong, and D. Fyhrie, “Comparison of the linear finite element prediction of deformation and strain of human cancellous bone to 3d digital volume correlation measurements,” *Journal of biomechanical engineering*, vol. 128, no. 1, pp. 1–6, 2006.
- [252] F. Forsberg, M. Sjö Dahl, R. Mooser, E. Hack, and P. Wyss, “Full three-dimensional strain measurements on wood exposed to three-point bending: analysis by use of digital volume correlation applied to synchrotron radiation micro-computed tomography image data,” *Strain*, vol. 46, no. 1, pp. 47–60, 2010.

- [253] S. Hall, M. Bornert, J. Desrues, Y. Pannier, N. Lenoir, G. Viggiani, and P. Bésuelle, “Discrete and continuum analysis of localised deformation in sand using x-ray μ ct and volumetric digital image correlation,” *Geotechnique*, vol. 60, no. 5, pp. 315–322, 2010.
- [254] B. Pan, D. Wu, and Z. Wang, “Internal displacement and strain measurement using digital volume correlation: a least-squares framework,” *Measurement Science and Technology*, vol. 23, no. 4, p. 045002, 2012.
- [255] B. Pan, B. Wang, D. Wu, and G. Lubineau, “An efficient and accurate 3d displacements tracking strategy for digital volume correlation,” *Optics and Lasers in Engineering*, vol. 58, pp. 126–135, 2014.
- [256] H. W. Schreier, J. R. Braasch, and M. A. Sutton, “Systematic errors in digital image correlation caused by intensity interpolation,” *Optical engineering*, vol. 39, no. 11, pp. 2915–2921, 2000.
- [257] B. Pan, “Bias error reduction of digital image correlation using gaussian pre-filtering,” *Optics and Lasers in Engineering*, vol. 51, no. 10, pp. 1161–1167, 2013.
- [258] Wikipedia, “Bicubic interpolation — wikipedia, the free encyclopedia,” 2017. [Online; accessed 27-September-2017].
- [259] Wikipedia, “Tricubic interpolation — wikipedia, the free encyclopedia,” 2017. [Online; accessed 27-September-2017].
- [260] M. Harris *et al.*, “Optimizing parallel reduction in cuda,” *NVIDIA Developer Technology*, vol. 2, no. 4, 2007.
- [261] C. Solutions, “Vic-2d v6 testing guide,” *Correlated Solutions. Irmo, SC, USA*, 2017.

- [262] F. Hild, A. Bouterf, P. Forquin, and S. Roux, “On the use of digital image correlation for the analysis of the dynamic behavior of materials,” in *The Micro-World Observed by Ultra High-Speed Cameras*, pp. 185–206, Springer, 2018.
- [263] M. S. Kirugulige, H. V. Tippur, and T. S. Denney, “Measurement of transient deformations using digital image correlation method and high-speed photography: application to dynamic fracture,” *Applied optics*, vol. 46, no. 22, pp. 5083–5096, 2007.
- [264] E. Li, A. Tieu, and W. Yuen, “Application of digital image correlation technique to dynamic measurement of the velocity field in the deformation zone in cold rolling,” *Optics and Lasers in Engineering*, vol. 39, no. 4, pp. 479–488, 2003.
- [265] Y. Zhou, B. Pan, and Y. Q. Chen, “Large deformation measurement using digital image correlation: a fully automated approach,” *Applied optics*, vol. 51, no. 31, pp. 7674–7683, 2012.
- [266] Y. Zhou and Y. Q. Chen, “Feature matching for automated and reliable initialization in three-dimensional digital image correlation,” *Optics and Lasers in Engineering*, vol. 51, no. 3, pp. 213–223, 2013.
- [267] R. Wu, H. Qian, and D. Zhang, “Robust full-field measurement considering rotation using digital image correlation,” *Measurement Science and Technology*, vol. 27, no. 10, p. 105002, 2016.
- [268] F. Zhong and C. Quan, “Digital image correlation in polar coordinate robust to a large rotation,” *Optics and Lasers in Engineering*, vol. 98, pp. 153–158, 2017.
- [269] Z. Tang, J. Liang, Z. Xiao, and C. Guo, “Large deformation measurement scheme for 3d digital image correlation method,” *Optics and Lasers in Engineering*, vol. 50, no. 2, pp. 122–130, 2012.

-
- [270] B. Pan, W. Dafang, and X. Yong, “Incremental calculation for large deformation measurement using reliability-guided digital image correlation,” *Optics and Lasers in Engineering*, vol. 50, no. 4, pp. 586–592, 2012.
- [271] B. Pan and L. Tian, “Advanced video extensometer for non-contact, real-time, high-accuracy strain measurement,” *Optics express*, vol. 24, no. 17, pp. 19082–19093, 2016.
- [272] J. Stam, “What every cuda programmer should know about opengl,” in *GPU Technology Conference*, 2009.
- [273] J.-J. Lee and M. Shinozuka, “Real-time displacement measurement of a flexible bridge using digital image processing techniques,” *Experimental mechanics*, vol. 46, no. 1, pp. 105–114, 2006.
- [274] D. McCracken, “Posix threads and the linux kernel,” in *Ottawa Linux Symposium*, p. 330, 2002.
- [275] L. Kai and Q. Kemaο, “Dynyamic 3d profiling with fringe projection using least squares method and windowed fourier filtering,” *Optics and Lasers in Engineering*, vol. 51, no. 1, pp. 1–7, 2013.
- [276] G. Wenjing, Q. Kemaο, W. Haixia, L. Feng, S. H. Soon, and C. L. Sing, “General structure for real time fringe pattern preprocessing and implementation of median filter and average filter on fpga,” in *Proc. SPIE*, vol. 7155, p. 71550Q, 2008.
- [277] W. Gao, N. T. T. Huyen, H. S. Loi, and Q. Kemaο, “Real-time 2d parallel windowed fourier transform for fringe pattern analysis using graphics processing unit,” *Optics express*, vol. 17, no. 25, pp. 23147–23152, 2009.
- [278] W. Gao and Q. Kemaο, “Real-time pipelined heterogeneous system for windowed fourier filtering and quality guided phase unwrapping algorithm using graphic processing unit,” in *AIP Conference Proceedings*, vol. 1236, pp. 129–134, AIP, 2010.

- [279] T. Wang, L. Kai, and Q. Kemao, “Real-time reference-based dynamic phase retrieval algorithm for optical measurement,” *Applied Optics*, vol. 56, no. 27, pp. 7726–7733, 2017.
- [280] B. N. Datta, *Numerical linear algebra and applications*. Siam, 2010.
- [281] M. Abdellah, “cufftshift: high performance cuda-accelerated fft-shift library,” in *Proceedings of the High Performance Computing Symposium*, p. 5, Society for Computer Simulation International, 2014.
- [282] R. Zhu, H. Xie, Z. Hu, L. Jiang, B. Guo, and C. Li, “Performances of different subset shapes and control points in subset-based digital image correlation and their applications in boundary deformation measurement,” *Applied optics*, vol. 54, no. 6, pp. 1290–1301, 2015.
- [283] L. P. Chew, “Constrained delaunay triangulations,” *Algorithmica*, vol. 4, no. 1-4, pp. 97–108, 1989.
- [284] S. Kleinendorst, J. Hoefnagels, C. Verhoosel, and A. Ruybalid, “On the use of adaptive refinement in isogeometric digital image correlation,” *International Journal for Numerical Methods in Engineering*, 2015.
- [285] B. Pan, B. Wang, G. Lubineau, and A. Moussawi, “Comparison of subset-based local and finite element-based global digital image correlation,” *Experimental Mechanics*, vol. 55, no. 5, pp. 887–901, 2015.

Publication

Journals

- Zhang, L., Wang, T., Jiang, Z., Kemaio, Q., Liu, Y., Liu, Z., ... & Dong, S. (2015). High accuracy digital image correlation powered by GPU-based parallel computing. *Optics and Lasers in Engineering*, 69, 7-12.
- Wang, T., Jiang, Z., Kemaio, Q., Lin, F., & HS, Seah. (2016). GPU accelerated digital volume correlation. *Experimental Mechanics*, 56(2), 297-309.
- Wang, T., & Qian, K. (2017). Parallel computing in experimental mechanics and optical measurement: A review (II). *Optics and Lasers in Engineering*.
- Wang, T., Kai, L., & Kemaio, Q. (2017). Real-time reference-based dynamic phase retrieval algorithm for optical measurement. *Applied Optics*, 56(27), 7726-7733.
- Wang, T., Kemaio, Q., HS, Seah., Feng, L (2018). A flexible heterogeneous real-time digital image correlation system, in preparation.

Book

- Wang, T., Kemaio, Q., (2018). *GPU Acceleration for Optical Measurement*, Bellingham, Wash, USA: SPIE Press.

Conference papers & presentations

- Wang, T., Kemaio. Q. (2016, July). Real-time fast Fourier transform-based digital image correlation. Paper presented at 17th International Conference on Experimental Mechanics, Rhodes, Greece.

REFERENCES

- Wang, T., Zhang, L., Jiang, Z., & Qian, K. (2017). GPU Accelerated High Accuracy Digital Volume Correlation. In International Digital Imaging Correlation Society (pp. 25-27). Springer, Cham.
- Wang, T., Kemao, Q. (2017, Apr). CPU and GPU Parallel Computing in Photomechanics Applications: A Review. Paper presented at International Conference on Optical and Photonic Engineering 2017, Expo, Singapore.
- Wang, T., Kemao, Q. (2017, April). Real-time Dynamic Phase Retrieval. Paper presented at International Conference on Optical and Photonic Engineering 2017, Expo, Singapore.