A high-speed accurate system for phase denoising and unwrapping

Gao, Wenjing

2012

Gao, W. (2012). A high-speed accurate system for phase denoising and unwrapping.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/50787>

<https://doi.org/10.32657/10356/50787>

Nanyang Technological University

A high-speed accurate system for phase denoising and unwrapping

**A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of Doctor of Philosophy
Division of Computing Systems
School of Computer Engineering**

By

Gao Wenjing

November, 2011

Acknowledgments

I would like to express my deep gratitude to Prof. Qian Kemao who gave me the opportunity to complete this work under his esteemed guidance. Without his encouragement, guidance and support, I would not finish this work smoothly. Furthermore, his rigorous toward the research impressed me and will undoubtedly influence me in the future when I do research work by myself.

I would also like to acknowledge Wang Haixia being not only the colleague of mine but also my close friend. During the past 4 years, we shared our insights on the related research topic as well as the available research resources.

In addition, a special thanks to all my friends in Singapore who gave me memorable happy times here. I am also indebted to my former friends, who unconditionally support me and believe me in my difficult times.

Last but not the least, I would like to dedicate this dissertation to my family for their love throughout my life.

Author's Publications

1. W. Gao, Q. Kemao, H. Wang, F. Lin, S. H. Soon, and L. S. Cheong, "General structure for real-time fringe pattern preprocessing and implementation of median filter and average filter on FPGA," Proc. SPIE, **7155** (2008).
2. Q. Kemao, W. Gao, and H. Wang, "Filtering-based phase unwrapping," Proc. SPIE, **7063** (2008).
3. Q. Kemao, W. Gao, and H. Wang, "Windowed Fourier-filtered and quality-guided phase unwrapping algorithm," Appl. Opt. **47**, 5420-5428 (2008).
4. H. Wang, Q. Kemao, and W. Gao, "Frequency-shifting method for phase retrieval from fringe patterns," Proc. SPIE, **7155** (2008).
5. Q. Kemao, H. Wang, and W. Gao, "Windowed Fourier transform for fringe pattern analysis: theoretical analyses," Appl. Opt. **47**, 5408-5419 (2008).
6. H. Wang, Q. Kemao, W. Gao, F. Lin, and S. H. Soon, "Partial differential equation based coherence enhancing diffusion for fringe patterns," International Conference on Experimental Mechanics (2008)
7. W. Gao, Q. Kemao, H. Wang, F. Lin, S. H. Soon, "Parallel computing for fringe pattern processing: A multicore CPU approach in MATLAB[®] environment," Opt. & Lasers in Eng. **47**, 1286-1292 (2009).
8. H. Wang, Q. Kemao, W. Gao, S. H. Soon, and F. Lin, "Fringe pattern denoising using coherence enhancing diffusion," Opt. Lett. **34**, 1141-1143 (2009).
9. W. Gao, N.T.T. Huyen, H.S. Loi, and Q. Kemao, "Real-time 2D parallel windowed Fourier transform for fringe pattern analysis using Graphics Processing Unit," Opt.

- Express **17**, 23147-23152 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-25-23147>.
10. W. Gao, and Q. Kemaio, "Real-time pipelined heterogeneous system for windowed Fourier filtering and quality guided phase unwrapping algorithm using Graphic Processing Unit," AIP Conference Proc. **1236**, 129-134 (2010).
 11. Q. Kemaio, W. Gao, and H. Wang, "Windowed Fourier filtered and quality guided phase unwrapping algorithm: on locally high-order polynomial phase," Appl. Opt. **49**, 1075-1079 (2010).
 12. Q. Kemaio, H. Wang, and W. Gao, "Some recent developments of windowed Fourier transform for fringe pattern analysis," AIP Conference Proc. **1236**, 106-111 (2010).
 13. W. Gao, and Q. Kemaio, "Phase compensation for the 2D windowed Fourier ridge algorithm," in CIOMP-OSA international summer school : Lasers and their applications, Changchun, China, 2011
 14. W. Gao, and Q. Kemaio, "Parallel computing in experimental mechanics and optical measurement: A review," Opt. & Lasers in Eng. **50**, 608-617(2012).
 15. W. Gao, and Q. Kemaio, "Statistical analysis for windowed Fourier ridges and phase compensation," Appl. Opt. **51**, 328-337(2012).
 16. W. Gao, and Q. Kemaio, "Statistical analysis of 2D windowed Fourier ridges algorithm," in *5th International Conference on Optical Measurement Techniques for Structures & Systems*, (4 & 5 April 2012 - Antwerp, Belgium, 2012).

Abbreviation and Symbol

2-D	Two-dimensional
3-D	Three-dimensional
4-D	Four-dimensional
ACPAS	Accurate compensated phase-added stereogram
ASIC	Application specific integrated circuit
Cell BE	Cell broadband engine
CGH	Computer generated hologram
CPF	Cubic phase function
CPU	Central processing unit
CRB	Cramér-Rao bounds
CUDA	Computer unified device architecture
DCC	Dynamic cost cycle canceling
DCT	Discrete cosine transform
DH	Digital holography
DHI	Digital holographic interferometry
DSP	Digital signal processor
EPF	Exponential phase field
ESPI	Electronic Speckle Pattern Interferometry
FFT	Fast Fourier transform
flops	Floating-point operations per second
FPGA	Field programmable gate array

fps	frame rates per second
GPU	Graphic processing unit
GPGPU	General-Purpose computation in GPUs
IFR	Instantaneous frequency rate
IP	Intellectual property
InSAR	Interferometric synthetic aperture radar
InSAS	Interferometric synthetic aperture sonar
LUT	Loop-up table
MIMD	Multiple instruction multiple data
MISD	Multiple instruction single data
MLE	Maximum likelihood estimation
MRI	Magnetic resonance imaging
MST	Minimum spanning tree
MSE	Mean squared error
PC	Personal computer
PCI	Peripheral component interconnect
PPE	Power processor element
PPT	Phase difference transform
PU	Phase unwrapping
RPT	Regular phase tracking
SGI	Silicon Graphics Inc.
SIMD	Single instruction multiple data
SIMT	Single instruction multiple thread
SISD	Single instruction single data

SMPs	Shared memory multiprocessors
SoC	System on a chip
SP	Single precision
SPE	Synergistic processing element
SNR	Signal to noise ratio
WFF	Windowed Fourier filtering
WFF-QG	Quality-guided phase unwrapping with the windowed Fourier filtering
WFR	Windowed Fourier ridges
WFR-QG	Quality-guided phase unwrapping with the windowed Fourier ridges
WFRC	Windowed Fourier ridges with phase compensation
WFRC-QG	Quality-guided phase unwrapping with the windowed Fourier ridges and phase compensation
WFT	Windowed Fourier transform
WR	Wavelet ridges
VLIW	Very long instruction word

Contents

Acknowledgments	I
Author's Publications	II
Abbreviation and Symbol	IV
Contents	VII
List of Figures	XII
List of Tables	XIV
Abstract	XV
Chapter 1 Introduction	1
1.1 High-speed 2-D phase denoising and unwrapping for fringe pattern analysis	1
1.2 Contributions and the outline of the dissertation	6
Chapter 2 Literature Review	11
2.1 Popular phase unwrapping algorithms	11
2.1.1 Path-following phase unwrapping algorithms	12
2.1.2 The minimum L^P -norm algorithms	15
2.1.3 The Bayesian/Regularization algorithms	17
2.1.4 The phase unwrapping algorithms based on parametric models	20
2.2 The windowed Fourier transform	25
2.3 Cramér–Rao bound and Maximum likelihood estimation	26
2.3.1 Cramér–Rao bound	26
2.3.2 Maximum likelihood estimation	27
2.3.3 Perturbation analysis preliminaries ^[76]	28

2.4 Parallel computing in experimental mechanics and optical measurement	29
2.4.1 Introduction.....	29
2.4.2 Parallel Computing	32
2.4.2.1 Parallel Methods	32
2.4.2.2 Amdahl's Law ^[81]	33
2.4.2.3 Hardware Platforms	34
2.4.2.4 Pros and cons of parallel hardware platforms.....	39
2.4.3 Applications in experimental mechanics and optical measurement: platforms based on MIMD.....	43
2.4.3.1 Computer clusters	43
2.4.3.2 Shared memory multiprocessors (SMPs)	44
2.4.3.3 General-purpose multicore processors.....	45
2.4.3.4 Pipeline processors	46
2.4.4 Applications in experimental mechanics and optical measurement: platform based on SIMD.....	46
2.4.4.1 SIMD parallel processors.....	46
2.4.4.2 Graphics processing units (GPUs)	47
2.4.5 Applications in experimental mechanics and optical measurement: platforms based on hybrid architectures	50
2.4.5.1 PC with a multicore CPU and GPUs	50
2.4.5.2 SGI Onyx computers	50

2.4.5.3 Digital signal processors (DSPs)	51
2.4.5.4 Application specific integrated circuits (ASICs)	53
2.4.5.5 Field Programmable Gate Arrays (FPGAs)	54
2.4.6 Discussions and conclusions.....	56
2.5 Conclusions.....	58
Chapter 3 Phase compensation for the WFR algorithm.....	60
3.1 Introduction.....	60
3.2 2-D windowed Fourier ridges algorithm	61
3.3 $\hat{c}_{20}$ and $\hat{c}_{02}$ estimates by linear equations	63
3.4 $\hat{c}_{20}$ and $\hat{c}_{02}$ estimates by least squares fitting	63
3.5 Verifications.....	65
3.5.1 For simulated wrapped phase map with Gaussian noise	67
3.5.2 For simulated wrapped phase map with speckle noise	69
3.5.3 For a simulated wrapped phase map with long discontinuities	70
3.5.4 For an experimental wrapped phase map obtained from fringe projection profilometry	72
3.6 Conclusion	74
Chapter 4 Statistical analysis for windowed Fourier ridges algorithm in fringe pattern analysis	76
4.1 Introduction.....	76
4.2 1-D windowed Fourier ridges algorithm	77
4.3 Estimation of local frequency and phase from a noisy signal	80
4.3.1 Local frequency estimation.....	80

4.3.2 Phase estimation	83
4.3.3 Comparison with the Cramér-Rao Bounds	89
4.4 Simulations	91
4.4.1 Monte Carlo simulations and comparison with CRBs.....	92
4.4.2 Influences of c_2 and window truncation	94
4.4.3 WFRF vs. WFF.....	96
4.4.4 Comparing with other model based phase estimation methods.....	97
4.5 Conclusion & Discussions	98
Chapter 5 Accelerating the WFT-based algorithms using multicore CPU in MATLAB [®] environment	
5.1 Introduction.....	100
5.2 MATLAB [®] parallel computing toolbox	102
5.3 Using parallel computing toolbox and multicore CPU to accelerate the WFT- based algorithms	105
5.3.1 Windowed Fourier transform based algorithms.....	105
5.3.2 Accelerating the WFF algorithm.....	107
5.3.3 Accelerating the WFR algorithm	109
5.4 Results and discussions.....	111
5.5 Conclusion	115
Chapter 6 Real-time 2-D parallel windowed Fourier transform for fringe pattern analysis using Graphic Processing Unit	
6.1 Introduction.....	116
6.2 Parallelization principles for windowed Fourier transform based algorithms...	118

6.3 Graphics computing Unit and computer unified device architecture	120
6.4 Parallel WFF and WFR algorithm based on CUDA.....	122
6.5 Experiments and performance of parallel WFF and WFR using CUDA and GPU	125
6.6 Conclusions and discussions.....	128
Chapter 7 A real-time pipelined heterogeneous system for WFT-based quality-guided PU algorithms	129
7.1 Introduction.....	129
7.2 Windowed Fourier transform based quality-guided PU algorithm.....	131
7.3 Real-time pipelined heterogeneous system based on multicore CPU and GPU	132
7.4 Experiments and Performance of the proposed system	136
7.5 Conclusion and discussions	139
Chapter 8 Conclusions	141
8.1 Summary	141
8.2 Future works	146
Bibliography	148

List of Figures

Figure 1.1 Phase retrieval in a digital holographic microscopy system	2
Figure 2.1 Phase residue	13
Figure 2.2 The mechanism of parallelism methods	31
Figure 2.3 Distribution of the reviewed literatures	57
Figure 3.1 Testing on an simulated wrapped phase map with quadratic phase and Gaussian noise	66
Figure 3.2 Testing on an simulated wrapped phase map with cubic phase and Gaussian noise	68
Figure 3.3 Testing on an simulated wrapped phase map with quadratic phase and speckle noise	70
Figure 3.4 Testing on an simulated wrapped phase map with 3dB noise	71
Figure 3.5 Testing on a experimental wrapped phase map from FPP	73
Figure 4.1 Simulated wrapped phase maps.	92
Figure 4.2 Theoretical and experimental MSEs by the WFRC algorithm.....	92
Figure 4.3 The influence of c_2 in the WFRC algorithm.....	94
Figure 4.4 The influence of window truncation, $6\sigma+1=61$ vs. $10\sigma+1=101$	95
Figure 4.5 Phase estimate by the WFRC and WFF algorithms	96
Figure 4.6 Comparison of phase estimate by the WFRC algorithm, HAF and CPF algorithms	98
Figure 5.1 Architecture using parallel computing toolbox and multicore CPU	103

Figure 5.2 MATLAB [®] code for sequential and task-parallel approach on WFF algorithm	107
Figure 5.3 Pseudo code for data-parallel wrapping function.....	110
Figure 5.4 Division and synthesis process for data-parallel approach.....	111
Figure 5.5 WFF Results using task-parallel and data-parallel approach	112
Figure 5.6 Speedup using proposed parallelism approaches on WFF.....	113
Figure 6.1 Pseudo codes of the parallel WFT library	123
Figure 6.2 Sequential and parallel WFF result for processing a digital holographic fringe pattern.	126
Figure 7.1 Mechanism of real-time pipelined heterogeneous system based on CPU and GPU	133
Figure 7.2 Pipelined WFF-QG algorithm based on multicore CPU and GPU	133
Figure 7.3 One frame processed by WFF-QG algorithm	138

List of Tables

Table 2.1 Comparison among processor-level hardware platform	42
Table 4.1 Theoretical MSEs of local frequency and phase estimates by the WFR and their CRBs.....	91
Table 6.1 Execution time for different size of fringe pattern by WFF and WFR algorithm..	127
Table 7.1 Frame rate for different size of wrapped phase by WFF-QG algorithm in different system.....	139

Abstract

Two-dimensional (2-D) fringe pattern analysis algorithms have been developed for several decades. They impel the frontier interferometric techniques such as synthetic aperture radar, interferometric synthetic aperture sonar, magnetic resonance imaging, diffraction tomography and optical measurement being widely used in more and more measurement applications. Due to the factors such as noises caused by acquisition mechanism and speckle effect, subsampling, and long discontinuities, the modern 2-D fringe analysis algorithms for phase denoising and unwrapping are complicated and usually time consuming. The windowed Fourier transform (WFT) based algorithms including the windowed Fourier filtering (WFF) and the windowed Fourier ridges (WFR) are among the effective fringe analysis algorithms. Along with quality-guided phase unwrapping algorithm, the WFT based algorithms can successfully reconstruct the phase by removing the noises and effectively identify the long discontinuities in the fringe patterns.

Despite the merits of the WFR algorithm, a small phase bias is produced by the WFR algorithm for phase estimate. In this dissertation, phase compensation methods are proposed for the WFR algorithm, which is denoted as the WFRC algorithm. Then the theoretical MSEs for local frequency and phase estimates by the WFRC algorithm are derived in the first order perturbation perspective and compared with the corresponding Cramér-Rao bounds to show the efficiency of the proposed algorithm. Another drawback of the WFT-based algorithms is their long computation time, prohibiting them from real-

time applications. However, the WFT-based algorithms are highly parallelizable, which implies the potential of being efficiently accelerated by the modern parallel computing hardware. Thus in this thesis we explore using parallel hardware to accelerate the WFT-based algorithms and propose a high-speed heterogeneous system based on multicore CPU and graphics processing units for phase denoising and unwrapping using the two algorithms. The system can be easily integrated into a fringe analysis system.

Key Words: Real time, Fringe analysis, Phase unwrapping, Windowed Fourier ridges, Cramér-Rao bounds, GPU.

Chapter 1 Introduction

1.1 High-speed 2-D phase denoising and unwrapping for fringe pattern analysis

In interferometric systems such as synthetic aperture radar (InSAR) [1], interferometric synthetic aperture sonar (InSAS) [2], magnetic resonance imaging (MRI) [3, 4], diffraction tomography [5] and optical measurement [6], coherently generated images are called fringe patterns or interferograms [7]. The retrieval of the continuous phase and corresponding physical information from incomplete, noisy fringe patterns is the task of fringe pattern analysis. Thanks to the techniques of fringe analysis developed during the past few decades, InSAR has turned from a pioneer technology into an accepted remote-sensing tool for applications such as the measurement of land terrain, ice-stream motion, seismic strain and land subsidence [8]; InSAS is employed as an innovative technique in underwater imaging to provide full resolution bathymetric maps [9]; MRI and diffraction tomography are widely used in medical diagnosis and cell imaging [4, 10, 11]; and optical measurement has been shown as a nondestructive, high-speed measurement tool for shape, displacement, deformation and vibration [6].

In fringe pattern analysis, one of the key steps is phase unwrapping (PU) by retrieving a continuous phase from a exponential phase field (EPF) which can be represented as follows,

$$f(x, y) = b(x, y) \exp[j\varphi(x, y)], \quad (1.1)$$

where $b(x, y)$ and $\varphi(x, y)$ are fringe amplitude and phase. To retrieve the continuous phase $\varphi(x, y)$ from $f(x, y)$, an arctangent function is used. However, the retrieved phase is wrapped and the value of the phase ranges from $-\pi$ to π . The correction of these 2π jumps to generate a continuous phase distribution is called phase unwrapping. Figures 1.1 show the measurement results of a laser mark on glass substrates produced by a digital holographic microscopy system in transmission mode [12]. The real part of resulting EPF, its wrapped phase map by arctangent function, and the retrieved three-dimensional (3-D) phase distribution $\varphi(x, y)$ are shown in Figs 1.1 (a-c), respectively.

Figure 1.1 Phase retrieval in a digital holographic microscopy system (a) Real part of an EPF (b) The wrapped phase map of the EPF by arctangent function (c) The 3-D profile of reconstructed phase

Usually, noises caused by the acquisition mechanism and speckle effect [6], under sampling, and long discontinuities in the fringe patterns make the phase unwrapping a non-trivial task [7]. Consequently, many efforts have been put into the development of 2-D PU algorithms aiming at an accurate solution which can reconstruct the true phase with low mean squared error (MSE) in spite of the noises, under sampling and long discontinuities. Classical PU methods are path-following methods and minimum-norm methods [7]. During the past decade, many new 2-D fringe analysis techniques emerged, which can be classified as the Bayesian/regularization based PU methods [13-22] and parametric model based PU methods [23-35]. Sometimes denoising is applied to fringe patterns before the PU process to improve the accuracy using the conventional digital image processing algorithms [7]. In fact, some of the PU algorithms incorporate denoising process to achieve reliable unwrapped results [13-22, 27-34]. So far, there is no universal phase unwrapping method for all the applications mentioned in the first paragraph. The algorithms are compared in different applications for users to choose a suitable method for their case [36-40]. In this thesis, we mainly focus on the fringe patterns in optical measurement.

By assuming the local phase field to be a 2-D quadratic polynomial, the windowed Fourier filtering (WFF) and the windowed Fourier ridges (WFR) [41] algorithms were developed for fringe pattern analysis. As pre-processes for PU algorithms, they were considered as promising tools among the spatial techniques in use for phase measurement [42]. The WFF algorithm has been successfully used in the measurement of 3-D profiles [43], thickness of thin films [44], deformation of a force plate [45] and a dragonfly's wings [46], refractive index [47], and vibration [48]. Furthermore, an effective PU

method was proposed recently based on the quality-guided phase unwrapping with WFF (WFF-QG) algorithm for both quadratic and higher-order polynomial phase [27, 35]. The WFF algorithm was able to suppress the noises effectively while maintained the useful information for successive phase unwrapping.

On the other hand, it was shown in [39] that the WFR algorithm has strong linkage with other popular fringe analysis methods such as the regular phase tracking (RPT) method without the regularization term, the multiple-parameter least-squares method, the autocorrelation function method, and the sine/cosine average filter. In the work of Huang et al. [49], detailed comparison was given among the Fourier transform, WFF, WFR, and wavelet transform algorithms for phase extraction from a single fringe pattern in phase projection profilometry, in which the 2-D WFR was shown to be more accurate than other algorithms for a continuous phase. In [27, 35], the quality-guided phase unwrapping with the WFR (WFR-QG) algorithm was also proposed. However, due to the bias produced by the WFR algorithm, the WFR-QG produced larger MSE than that of the WFF-QG algorithm [27]. Furthermore, since the WFR algorithm is based on a parametric model, its accuracy was discussed in [50]. However,

(i) The analysis of the local frequency estimate in Ref [50] was qualitative and a typical fringe pattern was utilized;

(ii) It was found that the phase estimate was biased [50] and needed to be compensated. But the method for phase compensation and the statistical analysis after phase compensation was not provided;

(iii) The efficiency of the WFR algorithm over the Cramér–Rao bound (CRB) was lacking in [50].

Although to be accurate is the prime goal of the PU algorithms, speed is another important issue concerning the development of those techniques. In a data processing system, real-time usually indicates that the system can provide data quickly enough to affect the progress of fieldwork or feed data timely to the decision-making process [51]. Thus in different system, the frame rate of the real-time process may be different. Sometimes, real-time phase retrieval is required in applications such as high-speed surface deformation and combustion. Many systems have demonstrated their potential in real-time applications [52-56]. However, they restricted themselves to simple fringe analysis algorithms, by which phase errors may occur for fringe patterns with high level of noise. On the other hand, the current trend in computer industry is to use multiple parallel processing units instead of frequency scaling technique. One can no longer count on a modern computer to improve the speed of his or her algorithm automatically, at least in the next few decades [57]. That is to say, only the algorithms that possess intrinsic parallel characters can be accelerated by parallelizing techniques in the new multicore CPU architecture. Thus parallelizability will be another factor to evaluate the efficiency of an algorithm.

Despite the merits of the WFF and the WFR algorithms, the computation time for both the algorithms is very long. Although many efforts have been put into a fast implementation of both the algorithms [58], the speedup is limited and still far from real time. Fortunately,

the WFT based algorithms are highly parallelizable, and it is possible that the computation speed rises dramatically by using modern parallel hardware without sacrificing the accuracy.

1.2 Contributions and the outline of the dissertation

In this thesis, the WFR algorithm with phase compensation (WFRC) together with quality-guided PU algorithm is proposed and denoted as the WFRC-QG algorithm. The high-speed implementations of the WFRC-QG algorithm and its sister algorithm the WFF-QG for phase unwrapping are followed. As we mentioned in Section 1.1, the frame rate of 'real-time' may vary among different systems. Thus in the thesis, we use the term 'high-speed' instead of 'real-time' to show the potential of a system in real-time applications. Within this theme, the main contributions of the dissertation can be summarized as follows,

1. Two methods are proposed to compensate the phase bias for the WFR algorithm, which is denoted as the WFRC algorithm;
2. Detailed statistical analyses for the 1-D WFRC algorithm are derived for the frequency and phase estimate. The deduced theoretical MSEs are compared with the corresponding standard CRBs and verified by the Monte Carlo simulations;

3. A new parametric model based phase unwrapping algorithm: the WFRC-QG is introduced;
4. The MSE of the WFRC-QG algorithm for phase estimate is compared with that of other parametric model based PU algorithms as well as the WFF-QG algorithm;
5. To select a proper hardware for accelerating the WFT based algorithms, parallel hardware platforms are surveyed as well as the literatures in the areas of experimental mechanics and optical measurement;
6. Acceleration of the WFF and WFR algorithm by using MATLAB[®] parallel computing toolbox [59] and multicore central processing unit (CPU) is explored as a prototype for the acceleration in graphics processing unit (GPU);
7. Near real-time WFF and WFRC algorithms are obtained by using GPUs on a commodity graphics card;
8. A high-speed heterogeneous system is proposed using the WFF-QG and WFRC-QG algorithms for phase denoising and unwrapping, which can be easily integrated into a fringe analysis system.

The dissertation is organized as follows:

In Chapter 2, a literature survey on the popular PU algorithms is presented in Section 2.1 including the path-following PU algorithms, the minimum L^P -norm algorithms, the Bayesian/regularization algorithms and the PU algorithms based on parametric models. In Section 2.2, the basic formulas of the windowed Fourier transform are introduced to serve as a background for later Chapters. Given the fact that the accuracy of the PU algorithms based on parametric models are bounded by the CRB, the CRB and maximum likelihood estimation (MLE) are introduced in Section 2.3. Moreover, the preliminary of first order perturbation analysis is also included in this section. At last, the literatures using parallel hardware are reviewed by Section 2.4 in the areas of experimental mechanics and optical measurement. The definition and background of parallel computing are also addressed in this section. Furthermore, the popular parallel hardware platforms are compared in the aspects of parallel type of the problem, performance per cost, energy consumption, and the development cycle. The conclusion of this Chapter is given in Section 2.5.

In Chapter 3, we propose two methods to compensate the phase bias produced by the WFR algorithm. Experimental results show that the proposed 2-D WFRC algorithm is resistant to the noises and performs better than the WFF algorithm for simulated fringe patterns. It is also shown by a case of experimental fringe pattern that the WFRC algorithm can successfully suppress the phase bias in the WFR algorithm and the accuracy for the phase reconstruction is improved. Also in this section, the WFRC-QG algorithm is proposed. Due to the high sensitivity to the edges and boundaries, the quality map produced by the WFRC-QG algorithm can effectively unwrap some fringe patterns

with long discontinuities which remains to be a difficult problem in phase unwrapping nowadays.

Chapter 4 deals with the statistical analysis of the WFRC algorithm. By assuming exponential phase field to be disturbed by additive white Gaussian noise, the MSEs for the phase and local frequency estimates are derived using the first order perturbation technique. Then the derived MSEs are compared with the corresponding CRBs theoretically and verified by the Monte Carlo simulations. Since the frequency estimation along the x and y axis are separable, we mainly focus on the derivation of the 1-D WFRC algorithm while its extension to the 2-D WFRC algorithm is straightforward. The WFRC algorithm is compared with its sister algorithm WFF and other algorithms for phase estimation to show its effectiveness.

In Chapter 5, multicore CPU and MATLAB[®] parallel computing toolbox are explored to accelerate the WFT based algorithms for fringe pattern analysis. As mentioned in Section 1.1, multicore has been developed into the mainstream technology of computer structure for both general processor and application specific integrated circuits (ASICs). However, a problem emerges that the former sequential algorithms cannot automatically adapt themselves into the new computer architecture. Furthermore, the conversion of sequential programs into their parallel versions is not a easy task. MATLAB[®] is among the first batch of software that supports programming multicore CPU. Several times speedup can be achieved with simple modification of the former MATLAB[®] codes. Although the speedup is not significant, because of the concise trait of MATLAB[®] codes, the work in

this Chapter can be used as a prototype for parallelizing the WFF and WFR algorithms in the GPU.

In Chapter 6, we propose a fast parallel WFT library using GPU and computer unified device architecture (CUDA) to accelerate the WFF and the WFR algorithms. Although the WFF and the WFR algorithms are time consuming in conventional sequential computers, they are highly parallelizable and can be efficiently accelerated using data parallel hardware. With massive parallel stream processors, GPU is a promising hardware which is cost effective, platform-independent [60], and ready to use in nearly all personal computers (PCs). With computer unified device architecture, GPU can be used as a coprocessor for CPU to accelerate the data-parallel time-consuming parts of an algorithm. Consequently, near real-time parallel WFT-based algorithms are achieved using multiple GPUs on a graphics card.

In Chapter 7, a high-speed pipelined heterogeneous system for the WFF-QG and WFRC-QG algorithms is obtained based on a multicore CPU and a GPU. The GPU is in charge of highly data-parallel WFF and WFRC algorithms while the multicore CPU executes quality-guided PU algorithm simultaneously but on different fringe frames. These processes are overlapped along the time axis, which is called the pipeline parallelism. Using a NVIDIA GTX285 graphics card and a quad-core 2.5GHz Intel(R) Xeon(R) CPU E5420, 2.15 fps can be achieved for the WFF-QG algorithm with fringes size 256×256 . The proposed high speed system will gain more speedup performance with the fast development pace of GPUs and real-time implementation is possible. The dissertation concludes in chapter 8.

Chapter 2 Literature Review

2.1 Popular phase unwrapping algorithms

To reconstruct the phase $\varphi(x, y)$ from $f(x, y)$ in Eq. (1.1), an arctangent function is used and the wrapped phase map $\psi(x, y)$ is retrieved as,

$$\psi(x, y) = \arctan 2[\Im(f(x, y)), \Re(f(x, y))], \quad (2.1)$$

where $\Re$ and $\Im$ denote the real and imaginary parts of $f(x, y)$, respectively. Since the “arctan2” function gives the values in the range $(-\pi, \pi]$, the unwrapping process should be applied to $\psi(x, y)$ to reconstruct a continuous phase field. The 2-D PU process can be defined as: given the wrapped phase $\psi(x, y) \in (-\pi, \pi]$, find the true phase $\varphi(x, y)$, which satisfies

$$\psi(x, y) = W[\varphi(x, y)] = \varphi(x, y) + 2\pi k(x, y), \quad (2.2)$$

where $W[\cdot]$ is the wrapping operator, $k(x, y)$ is an integer function that enables $\psi(x, y) \in (-\pi, \pi]$. The 2-D phase unwrapping is widely considered as a very difficult problem [7]. Due to the noises, under sampling (aliasing) and true phase discontinuities on the wrapped phase maps, the simple line-by-line unwrapping technique by Itoh [7] often leads to failure for the phase reconstruction. The state-of-the-art 2-D PU algorithm can be classified into four categories: the path-following PU algorithms, the minimum L^P -norm algorithms, the Bayesian/Regularization algorithms, and the PU algorithms based

on parametric models. The first two categories have been thoroughly discussed by Ghiglia and Pritt in [7]. In the following subsections, we will briefly introduce the first two categories, enrich them with the new algorithms from 1998 to the present, and review the other two categories of PU algorithms which were mostly introduced recently.

2.1.1 Path-following phase unwrapping algorithms

In path-following PU algorithms, to avoid the corrupted areas on wrapped phase maps, the phase is unwrapped through a certain path. Typical path-following methods are Goldstein's branch cut algorithm [61], the quality-guided path-following algorithm [62], the mask cut algorithm [63, 64], the FLYNN's minimum discontinuity approach [65], the newly developed network based algorithms [66, 67] and the WFF-QG algorithm [27].

As a classical path-following approach, Goldstein's branch cut algorithm identifies the residues and balances them by connecting them with branch cuts that obstruct the unwrapping path. Figure 2.1 shows the mechanism of the phase residue and the placement of branch cuts. In Fig. 2.1(a), the wrapped phase map of pyramid is corrupted by invalid areas. Inconsistencies caused by noises or discontinuities on the wrapped phase map are detected by summing wrapped phase differences (shown in Eq. (2.3-2.6)) in a loop of every 2×2 window.

$$\Delta_1 = W \{ \psi(m, n+1) - \psi(m, n) \} \quad (2.3)$$

$$\Delta_2 = W \{ \psi(m+1, n+1) - \psi(m, n+1) \} \quad (2.4)$$

$$\Delta_3 = W\{\psi(m+1, n) - \psi(m+1, n+1)\} \quad (2.5)$$

$$\Delta_4 = W\{\psi(m, n) - \psi(m+1, n)\} \quad (2.6)$$

where W is the wrapping process, which adding $\pm 2k\pi$ to ensure the wrapped phase difference being $(-\pi, \pi]$.

Figure 2.1 Phase residue (a) The simulated wrapped phase map of corrupted pyramid with corrupted areas (b) The residue map of (a)

Residues $q = \sum_{i=1}^4 \Delta_i$ are flagged where the sum is nonzero and are classified as positive residues or negative residues according to the sign of the sum. The residue map of Fig 2.1(a) is shown in Fig 2.1(b) and one of the branch cuts (red line) is placed between the positive and negative residues which are shown as white and black points, respectively. After the branch cuts are placed and balanced, the phase can be unwrapped along any path that does not cross the branch cuts. The Goldstein's branch cut algorithm is very fast.

However, due to the near neighbor strategy [7], the branch cuts placed by the algorithm do not always indicate the right location of the contaminated pixels. Consequently, the entire region of the unwrapping phase may be affected by poorly placed branch cuts.

Being a region-growing algorithm, the quality-guided path-following algorithm unwraps the wrapped phase map from high quality pixels to low quality ones. The algorithm relies on a good quality map to achieve a correct result [68]. The mask cut algorithm combines the advantage of both Goldstein's branch cut algorithm and the quality-guided path-following algorithm using a quality map to guide the placement of branch cuts. Based on the minimum discontinuity principle, FLYNN's algorithm identifies the discontinuity wraps, and iteratively removes the miscounted fringes due to the noises. FLYNN's algorithm can achieve a global minimum and it is shown that the algorithm is guaranteed to terminate after a finite number of operations. However, FLYNN's algorithm also depends on a good quality map to achieve correct results for some wrapped phase maps.

Newly developed path-following algorithm by Costantini is based on a nonlinear minimization problem with integer variables [67]. It maps the PU problem to the problem of finding the minimum cost flow on a network, which is linear and can be solved by very efficient algorithms. With the constraint that the weighted deviation between the derivatives of the estimated and unknown phase must differ by integer multiples of 2π , the spread of errors is prevented. Moreover, the unwrapping results are less sensitive to slight changes of the weighting mask used. This process is to ensure that a global minimum cost flow can be found [67]. Follow the Costantini's work, Chen et al. [66] proposed two effective PU algorithms based on network approaches. Minimum spanning

tree (MST) uses the network idea of shortest paths and spanning trees to improve the Goldstein's algorithm. As fast as the Goldstein's branch cut algorithm, MST algorithm provides complete unwrapping with localized errors in the areas of dense residue, while the branch cut algorithm fails to provide a solution. The other is dynamic cost cycle canceling (DCC) which extends the ideas of linear network flow problems to the nonlinear L^0 case. Instead of exactly optimizing the objective of the problem, their efforts mainly focused on an approximation. Though the DCC algorithm outperforms the minimum L^P norm, the FLYNN's algorithm and the least squares algorithms, it is most computational expensive and requires comparable more memory space.

Another newly proposed PU algorithm was introduced in our previous work using the windowed Fourier filtering to improve the quality-guided PU algorithm [27, 35], which was denoted as the WFF-QG algorithm. The WFF algorithm is used before the quality-guided PU algorithm by providing a reliable quality map and a filtered wrapped phase to improve the success ratio of the quality-guided PU algorithm. Experimental results showed that the WFF-QG algorithm could unwrap the wrapped phase maps with noises and long discontinuities without relying on other quality maps.

2.1.2 The minimum L^P -norm algorithms

Minimum L^P -norm algorithms are based on optimization problems that minimize the differences between the gradients of the solution surface and the wrapped phase differences. Minimum L^P -norm algorithms include the unweighted least squares methods,

the weighted least squares methods, and the general minimum L^P -norm algorithm [7].

They can be generalized in Eq. (2.7),

$$L = \text{minimize} \left\{ \sum_{ij} w_{i,j}^{(x)} \left| \Delta \phi_{i,j}^{(x)} - \Delta_{i,j}^x \right|^p + \sum_{ij} w_{i,j}^{(y)} \left| \Delta \phi_{i,j}^{(y)} - \Delta_{i,j}^y \right|^p \right\}, \quad (2.7)$$

where $\Delta \phi^{(x)}$ and $\Delta \phi^{(y)}$ are the gradients of the unwrapped phase along x and y axis while $\Delta_{i,j}^x = W \{ \psi_{i+1,j} - \psi_{i,j} \}$ and $\Delta_{i,j}^y = W \{ \psi_{i,j+1} - \psi_{i,j} \}$ are the wrapped phase differences along x and y axis. Here, “ W ” is also a wrapping process. $w_{i,j}^{(x)}$ and $w_{i,j}^{(y)}$ are the user-defined weights along the x and y axes, respectively. Equation (2.7) is the general form of minimum L^P -norm algorithms. When $p=2$, Eq. (2.7) becomes the weighted least squares algorithms. When $p=2$ and $w_{i,j}^{(x)} = w_{i,j}^{(y)} = 1$, Eq. (2.7) is equivalent to the unweighted least squares algorithms.

Though faster than other minimum L^P -norm algorithms, the unweighted least squares algorithms unwrap through rather than around the residues. As a result, the residues may corrupt the unwrapped phase. The weighted least squares methods use predetermined weights to avoid integrating through the residues. They require a good quality map to define the weights. Many algorithms have been proposed to efficiently solve unweighted and weighted least squares problems: fast Fourier transform based unweighted least squares algorithm and multigrid algorithm for the unweighted least squares problem; weighted multigrid algorithm and preconditioned conjugate gradient for the weighted least squares problem, respectively. At last, the minimum L^P -norm algorithm treats the phase data in a global fashion and can be used with or without input quality map or mask [93]. It is a generalization of unweighted and weighted least squares phase unwrapping

algorithms and is the most computationally intensive among all above PU algorithms in this subsection.

2.1.3 The Bayesian/Regularization algorithms

Another important category of PU algorithms is based on regularization or Bayesian statistics. Due to the limited information provided by the fringe patterns caused by noises and corruption, additional constraints or prior knowledge to the original phase can be adopted to solve the PU problem. Different from the path-following and minimum L^P -norm algorithms which usually depend on precondition quality maps to guide the unwrapping process, algorithms in this section choose a suitable energy or cost function as prior information to constrain the search space of unwrapped functions. By Bayesian estimation perspective, the absolute phase φ can be obtained with maximum a posterior estimate [16] as,

$$\hat{\varphi} = \arg \max_{\varphi} \{P(\varphi, \psi)\}, \quad (2.8)$$

$$P(\psi, \varphi) = L(\psi|\varphi) + \lambda R(\varphi), \quad (2.9)$$

where term $L(\psi|\varphi)$ measures the faithfulness of the estimate to the data, and $R(\varphi)$ is a regularization term weighted by a hyperparameter λ .

Some Bayesian algorithms can be viewed as the generalization of the least squares algorithms. Marroquin et al. [13] first introduced regularization to the phase unwrapping and extended the least-squares integration of wrapped phase gradient by adding a

regularization term to the searched solution. Their algorithm could eliminate some noises in the unwrapped phase while interpolated over regions with missing information. However, the algorithm could not deal with the wrapped phase with high noise level and high phase gradients. Servin et al. [14] improved Marroquin's work by assuming a small region of unwrapped phase being smooth enough to be modeled by a plane. Furthermore, the plane must be close to the observed phase map in the wrapped space. By minimizing a cost function with the above two requirements, the RPT algorithm is capable of unwrapping low frequency regions as well as severely degraded regions on the wrapped phase maps. The main drawback of the RPT algorithm is the difficulty of local window selection.

Another regularization technique based on stochastic relaxation was proposed by Guerriero et al. [15]. Instead of assuming noises as Gaussian model and the magnitude of the true phase-field gradients being less than π everywhere, they formulated phase unwrapping as a constrained optimization problem for the field of integer multiples of 2π . Different from the above two regularization algorithms, this algorithm only removes the noises that lead to inconsistencies. Thus they obtained good solution for wrapped phase with severe noise and undersampling. In the work of Nico et al. [16], a theoretical study was conducted for the Bayesian approaches to solve the phase unwrapping problem. They discussed the Bayesian modeling theory, different kinds of regularization terms, and the previous PU algorithms based on Bayesian and regularization. They also generalized many popular PU algorithms such as the least squares and FLYNN's minimum discontinuity algorithms in the Bayesian perspective. Then they proposed a Bayesian algorithm based on the multiscale Gaussian Markov random fields starting from a single

resolution. Like Guerriero's work, they assumed that the adjacent phase samples do not necessarily differ less than π , which means their algorithm being resistant to the noises as well as alias. However, Guerriero's method and Nico's method were not able to unwrap the wrapped phase with long discontinuities.

Chen and Zebker [17] adopted generalized, non-linear cost functions for the probabilities of the wrapped phase, image intensity and interferogram in some particular applications. Their algorithm was capable of unwrapping regions with layover and long discontinuities in the absolute phase. However, the generalization was sacrificed and the applications should be modeled case by case. Recently, a series of PU algorithms were proposed based on Bayesian and energy minimization [18-21]. $Z\pi M$ algorithm [18] is based on a compound Gauss-Markov random field tailored to piecewise smooth absolute phase image with abrupt variations between neighboring regions. The algorithm consists of two steps within an iterative scheme. In each iteration, a discrete optimization step is used first and followed by an iterative conditional model step for smoothing. The algorithm was also compared with other popular PU algorithms to show its effectiveness. PUMA algorithm [19] was a generalized improved version of $Z\pi M$ based on the first-order Markov random fields. An exact energy minimization approach was proposed for convex potentials with the smooth phase surface and an approximate solution for nonconvex clique potentials with the layover and discontinuities on the true phase surface. Both the algorithms were solved by efficient graph cut techniques. PEALS algorithm [20] employed filtering based on local polynomial approximation and PUMA algorithm for PU step. It was shown more accurate than the previous algorithms for phase reconstruction. In the latest work, CAPE algorithm [21] was introduced to improve the

PUMA algorithm. PUMA was used first to unwrap the interferogram. Then denoising was applied if PUMA located the discontinuities. Similar work was conducted by Rivera and Marroquin [22] with Bayesian models for both convex and nonconvex case. Although very robust, one problem of the algorithms introduced in this paragraph such as CAPE [21] and Rivera's [22] algorithms is that different models have to be chosen for different type of fringe patterns, prohibiting them from real-time applications.

2.1.4 The phase unwrapping algorithms based on parametric models

Since the true phase can be expended by Taylor series of polynomial, $\varphi(x, y)$ can be assumed as a parametric model. Usually a polynomial function is used, and fringe patterns $f(x, y)$ can be represented by

$$f(x, y) = b(x, y) \exp[j\varphi(x, y)] + n(x, y), \quad (2.10)$$

$$\varphi(x, y) = \sum_{n=0}^N \sum_{m=0}^M c(x, y) x^n y^m, \quad (2.11)$$

where $c(x, y)$ are coefficients of the polynomial; M and N are the highest order of polynomial along the y and x axis, respectively; $b(x, y)$ is the amplitude of EPF and usually assumed to be constant b_0 ; and $n(x, y)$ is the additive noise. Then the modeled EPF is fitted to the observed fringe pattern on approximation basis. In parametric model based algorithms, the EPF $f(x, y)$ is usually used as input signal instead of the wrapped phase map.

Friedlander and Francos [23] first introduced phase unwrapping based on parametric model fitting. A 2-D polynomial model was used to fit the observed wrapped phase and a phase differencing algorithm was adopted to estimate the parametric model. Then the estimated phase was adopted as a reference to direct the actual PU process, which is much alike the congruence process [7]. In practice, the wrapped phase map was often segmented into regions and each one fitted its own polynomial model. Similar work was conducted by Liang [24] using polynomial model to fit the fringe patterns. However, they did not segment the fringe pattern, so it would introduce a large error when fitting a modeled polynomial EPF to the whole fringe pattern with high level of noise and discontinuities.

In the series of works proposed by Gorthi and Rastogi [29-34], the proposed algorithms were based on 1-D polynomial version of model in Eqs. (2.10) and (2.11). Thus an additional operation of phase stitching should be used to make the whole phase field continuous for a 2-D image. In [29], 1-D MLE method was used to solve the parametric fitting problem by assuming the piecewise phase to be quadratic polynomial model. It was seen that the MLE algorithm approached the CRB asymptotically. Although many searching techniques could be used to accelerate the algorithm, the computation time was still long. To accelerate the MLE method, they introduced many other suboptimal methods for phase unwrapping.

For digital holographic interferometer, 1-D phase difference transform (PPT) was used to estimate the parametric phase, which is called the high-ambiguous function (HAF) algorithm [30, 31]. The PPT operator is defined by the following formulas,

$$PT_M[f, \omega, \tau] = \sum_{x=(M-1)\tau}^{M_s} P_M[f(x), \tau] \exp(-j\omega x), \quad (2.12)$$

$$P_M[f(x), \tau] = \prod_{m=0}^{M-1} [f^{\dagger q}(x - m\tau)]^{\binom{M-1}{m}}, \quad (2.13)$$

where $f(x)$ is the input EPF signal whose phase is assumed to be a 1-D polynomial and

$$f^{\dagger q}(x) = \begin{cases} f(x) & \text{if } m \text{ is even} \\ f^*(x) & \text{if } m \text{ is odd} \end{cases}, (\cdot)^* \text{ denotes complex conjugation, } \tau \text{ is a positive real}$$

number and $\binom{x}{y} = \frac{x!}{(x-y)!y!}$, $M_s \times N_s$ is the dimension of the fringe pattern.

The PPT operator produces a signal tone signal and by identifying the peak in the FFT spectrum, the current highest order of the polynomial phase can be calculated using

$$c_M = \frac{\omega_0(\tau)}{M! \tau^{M-1}}, \quad (2.14)$$

where $\omega_0(\tau)$ is the maximum frequency of Eq. (2.12). Then the current highest order phase term can be removed by “peeling off” operations $f(x)' = f(x) \cdot \exp(-jc_M x^M)$, and $f(x)'$ is used for estimating the coefficient c_{M-1} in the next iteration until all the coefficients are estimated. Thus the original polynomial phase can be easily reconstructed using these coefficients. The proposed algorithm was compared with the Goldstein, the $Z\pi M$, and the WFR [41] algorithm by experiments. It was shown that the WFR and the HAF algorithms performed better than the other two algorithms, and the HAF algorithm

was slightly better than the WFR algorithm. Due to the fact that one polynomial is hard to fit the entire phase field when the phase is rapidly varying, a 1-D windowed version of phase difference algorithm was proposed [32, 33]. A simple window as well as a overlapping window methods were used to localize the signal fitting and overlapping window method was shown more robust than the simple window method. Furthermore, interpolation on Fourier coefficients was shown to improve the speed performance of HAF significantly.

Recently, 1-D cubic phase function (CPF) was adopted for fringe analysis [34]. In CPF algorithm, the phase is assumed to be a 1-D chirp signal with polynomial order of 3. The discrete CP function is defined as,

$$CP_f(x, \Omega) = \sum_{\tau=0}^{(N_w-1)/2} f(x+\tau)f(x-\tau)\exp(-j\Omega\tau^2), \quad (2.15)$$

where $f(x)$ is an EPF with the phase assumed to be 1-D polynomial with maximum order of 3, N_w is the size of the window or signal segment; τ is a real number and Ω is the frequency of the spectrum. It is derived by O'Shea [69] that the peak of CP function was equal to the second order derivative of the phase which was also denoted as instantaneous frequency rates (IFRs) $2(c_2 + 3c_3x)$. The CPF algorithm can be summarized as,

1) Estimate two IFRs $\hat{\Omega}_1$ and $\hat{\Omega}_2$ at times x_1 and x_2 , respectively

$$\hat{\Omega}_1 = \arg \max_{\Omega} |CP_f(x_1, \Omega)|. \quad (2.16)$$

$$\hat{\Omega}_2 = \arg \max_{\Omega} |CP_f(x_2, \Omega)|. \quad (2.17)$$

2) Let $\hat{c} = [\hat{c}_2 \quad \hat{c}_3]^T$, $\hat{R} = [\hat{\Omega}_1 \quad \hat{\Omega}_2]^T$ and $X = \begin{bmatrix} 2 & 6x_1 \\ 2 & 6x_2 \end{bmatrix}$. Then compute $\hat{c}$ by Eq. (2.18)

$$\hat{c} = X^{-1}\hat{R}. \quad (2.18)$$

3) Estimate $\hat{c}_1$ by dechirping and finding the Fourier transform peak

$$\hat{c}_1 = \arg \max_{c_1} \left| \sum_{x=-(N_w-1)/2}^{(N_w-1)/2} f(x) \exp \left[-j(\hat{c}_2 x^2 + \hat{c}_3 x^3) \right] \right|. \quad (2.19)$$

4) Find $\hat{b}_0$ and $\hat{c}_1$ by

$$\hat{b}_0 = \frac{1}{N_w} \left| \sum_{x=-(N_w-1)/2}^{(N_w-1)/2} f(x) \exp \left[-j(\hat{c}_1 x + \hat{c}_2 x^2 + \hat{c}_3 x^3) \right] \right|, \quad (2.20)$$

$$\hat{c}_0 = \text{angle} \left[\sum_{x=-(N_w-1)/2}^{(N_w-1)/2} f(x) \exp \left[-j(\hat{c}_1 x + \hat{c}_2 x^2 + \hat{c}_3 x^3) \right] \right]. \quad (2.21)$$

It was shown that the 1-D CPF algorithm was more robust than the HAF algorithm for chirp signal with low SNRs.

By assuming the phase field being locally 2-D quadratic polynomial, the WFR algorithm [41] and the wavelet ridges (WR) [25, 26, 70] algorithm can also be classified to this category. Although for other algorithms of this subsection, wrapped phase map can be unwrapped as well as smoothed simultaneously, the high frequencies and long discontinuities in the phase may introduce errors in those algorithms. Furthermore, the estimation of the lower order coefficients of the polynomial requires the estimation results of the higher order thus the errors can be easily accumulated. By contrast, the WFR algorithm and the WR algorithm smooth the EPF first by assuming the phase as locally low order polynomials. High frequencies and long discontinuities are easily identified by

the amplitude of filtering result [27, 28, 35] which can be used as a quality map in subsequent quality-guided PU algorithm.

2.2 The windowed Fourier transform

We introduce the basics of the windowed Fourier transform in this section for later Chapters. The windowed Fourier transform which is also denoted as short-time Fourier transform reveals the frequency change over space [41, 71]. A 2-D windowed Fourier transform pair can be expressed as,

$$Sf(u, v, \xi, \eta) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) g(x-u, y-v) e^{-j(\xi x + \eta y)} dx dy, \quad (2.22)$$

$$f(x, y) = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} Sf(u, v, \xi, \eta) g(x-u, y-v) e^{j(\xi x + \eta y)} d\xi d\eta du dv, \quad (2.23)$$

where $j = \sqrt{-1}$; $g(x-u, y-v)$ is a real and symmetric window function with finite energy and localizes the Fourier integral within the neighborhood of a point (u, v) , $f(x, y)$ is the input signal; $e^{j(\xi x + \eta y)}$ denotes a sinusoidal frequency modulation term, (ξ, η) are frequencies along the x and y axis respectively and frequency ranges are usually set as $\xi \in [-\pi, \pi]$, $\eta \in [-\pi, \pi]$ after discretization [72]. In Eq. (2.23), $f(x, y)$ is recovered from the windowed Fourier spectrum $Sf(u, v, \xi, \eta)$.

2.3 Cramér–Rao bound and Maximum likelihood estimation

2.3.1 Cramér–Rao bound

The MSE of a deterministic parameter by any unbiased estimator (of which MSE is equal to the variance) is bounded by the Cramér–Rao bound [73, 74]. For a parameter estimation, the ratio of the CRB to the corresponding MSEs by any unbiased estimator is called the efficiency of that estimator. An estimator that attains the CRB is called an efficient estimator. For an unbiased estimator, the CRB is defined as the inverse of the Fisher information as follows,

$$\text{Var}(\hat{\theta}) \geq \frac{1}{I(\theta)}, \quad (2.24)$$

where

$$I(\theta) = E \left\{ \left[\frac{\partial \ln p(x; \theta)}{\partial \theta} \right]^2 \right\} = -E \left[\frac{\partial^2 \ln p(x; \theta)}{\partial \theta^2} \right], \quad (2.25)$$

$p(x; \theta)$ is the PDF of measurements x for the deterministic parameter θ . $E[\]$ calculates the expected value of the term in the bracket throughout this thesis. The CRB can also be used to bound the variance of biased estimators of given bias as follows,

$$\text{Var}(\hat{\theta}) \geq \frac{[1 + b'(\theta)]^2}{I(\theta)}, \quad (2.26)$$

where $b(\theta)$ is the bias of the estimator with $b(\theta) = E(\hat{\theta}) - \theta$ and $b'(\theta)$ is the first order derivative of $b(\theta)$. It can be seen that Eq. (2.24) is a special case of Eq. (2.26) with $b(\theta) = 0$.

2.3.2 Maximum likelihood estimation

It is well known that the MLE attains the CRB asymptotically, which means it is an optimal solution [75]. If the phase is assumed to be a locally quadratic polynomial, the fringe patterns with noises can be modeled as,

$$f(x, y) = f_0(x, y) + n(x, y), \quad (2.27)$$

$$f_0(x, y) = b \exp[j\varphi(x, y)], \quad (2.28)$$

$$\varphi(x, y) = (c_{00} + c_{10}x + c_{01}y + 0.5c_{20}x^2 + 0.5c_{02}y^2), \quad (2.29)$$

where $f_0(x, y)$ is the EPF without noise, $n(x, y)$ is the assumed additive white Gaussian noise with zero mean and deviation σ_n for both x and y axis. b is the amplitude and assumed to be constant while $\varphi(x, y)$ is the phase expanded by the second order Taylor series. $c_{00} \sim c_{02}$ indicate the parameters of the polynomial model. It should be mentioned that the coefficient c_{11} is omitted since it is usually small in a small area. The instantaneous frequency (IF) of the phase along x and y axis are defined as,

$$\omega_x(x, y) = \frac{d\varphi(x, y)}{dx} = c_{10} + c_{20}x, \quad (2.30)$$

$$\omega_y(x, y) = \frac{d\varphi(x, y)}{dy} = c_{01} + c_{02}y, \quad (2.31)$$

in which $\omega_x(x, y)$ and $\frac{d(\cdot)}{dx}$ denote the local frequency and first order derivatives along the x axis while $\omega_y(x, y)$ and $\frac{d(\cdot)}{dy}$ are their y axis counterparts.

When MLE is applied to Eq. (2.27), the parameters $c_{00} \sim c_{02}$ should be estimated first and substituted back to the polynomial model to reconstruct the phase. The maximum likelihood estimator can be represented as,

$$(\hat{c}_{01}, \hat{c}_{02}, \hat{c}_{10}, \hat{c}_{20}) = \arg \max_{(c_{01}, c_{02}, c_{10}, c_{20})} \left| \sum_{y=0}^M \sum_{x=0}^N f(x, y) e^{-j(c_{01}y + c_{10}x + 0.5c_{02}y^2 + 0.5c_{20}x^2)} \right|, \quad (2.32)$$

$$\hat{b} = \frac{1}{MN} \left| \sum_{y=0}^M \sum_{x=0}^N f(x, y) e^{-j(\hat{c}_{01}y + \hat{c}_{10}x + 0.5\hat{c}_{02}y^2 + 0.5\hat{c}_{20}x^2)} \right|, \quad (2.33)$$

$$\hat{c}_{00} = \text{angle} \left[\sum_{y=0}^M \sum_{x=0}^N f(x, y) e^{-j(\hat{c}_{01}y + \hat{c}_{10}x + 0.5\hat{c}_{02}y^2 + 0.5\hat{c}_{20}x^2)} \right], \quad (2.34)$$

where $\hat{c}_{00} \sim \hat{c}_{02}$ are the estimated phase parameters and $\hat{b}$ is the estimated amplitude; $M \times N$ is the size of the fringe patterns. It can be seen from Eq. (2.32) that 4-D search is required, so the MLE algorithm is quite time consuming.

2.3.3 Perturbation analysis preliminaries^[76]

This section serves as preliminaries of first order perturbation which will be used in Chapter 4. Given a complex valued function $g_N(\Omega)$ that depends on a variable Ω and on the number of data points N , let

$$f_N(\Omega) = |g_N(\Omega)|^2 = g_N(\Omega)g_N^*(\Omega), \quad (2.35)$$

where the symbol $*$ denotes the conjugate of a complex number. Assume that $f_N(\Omega)$ has a global maximum at $\Omega = \Omega_0$. If $g_N(\Omega)$ is perturbed by $\delta g_N(\Omega)$, then the global maximum point will shift to $\Omega = \Omega_0 + \delta\Omega$. A first order approximation for $\delta\Omega$ is as follows,

$$\delta\Omega \approx -\frac{B}{A}, \quad (2.36)$$

where

$$A = 2 \operatorname{Re} \left[g_N(\Omega_0) \frac{\partial^2 g_N^*(\Omega_0)}{\partial \Omega^2} + \frac{\partial g_N(\Omega_0)}{\partial \Omega} \frac{\partial g_N^*(\Omega_0)}{\partial \Omega} \right], \quad (2.37)$$

$$B = 2 \operatorname{Re} \left[g_N(\Omega_0) \frac{\partial \delta g_N^*(\Omega_0)}{\partial \Omega} + \frac{\partial g_N(\Omega_0)}{\partial \Omega} \delta g_N^*(\Omega_0) \right]. \quad (2.38)$$

The MSE of $\delta\Omega$ is,

$$E[(\delta\Omega)^2] \approx \frac{E(B^2)}{A^2}. \quad (2.39)$$

2.4 Parallel computing in experimental mechanics and optical measurement

2.4.1 Introduction

As mentioned in chapter 1, the parallel computing hardware can be used to accelerate the WFT based algorithms for phase retrieval. Furthermore, nowadays the concept of parallel computing has permeated into personal computers (PCs) by using general-purpose multicore processors. Sequential algorithms should be modified, if possible, to achieve higher performance in those computers equipped with powerful multicore processors. Thus it is useful to study the mechanism of parallel computing.

In this section, to accelerate the WFT based algorithms, we explore the principle of parallel computing and survey the popular parallel computing hardware. Although there are applications of parallel computing in the areas such as InSAR, InSAS, MRI and diffraction tomography, we restrict ourselves to the literatures in experimental mechanics and optical measurement and take them as examples to show the characteristic of different parallel hardware platforms. Main principles of parallel computing at an application level as well as hardware platforms that support parallel computing are introduced in Section 2.4.2. In Sections 2.4.3 to 2.4.5, we review the works using parallel hardware in experimental mechanics and optical measurement. The discussions and conclusions are given in Section 2.4.6.

Figure 2.2 The mechanism of parallelism methods (a) Task parallelism (b) Pipeline parallelism (c)

Data parallelism. Reprint from Ref. [77] with permission from Elsevier.

2.4.2 Parallel Computing

2.4.2.1 Parallel Methods

By definition, parallel computing targets at solving a problem faster by dividing the problem into a set of sub-problems which are simultaneously executed by multiple independent computing units or nodes [78]. The commonly used parallelism methods include task parallelism, pipeline parallelism, and data parallelism [79, 80]. Most problems are combinations of the above three parallelism types.

Task parallelism distributes tasks across different parallel computing nodes. Illustrated in Fig. 2.2(a), tasks are implemented as threads, then each thread is executed in a different node to process the same or different data at the same time.

Pipeline parallelism can be viewed as a particular case of task parallelism where multiple tasks depend on each other and process different data simultaneously. Taken the case in Fig. 2.2(b) as an example, suppose there are three tasks executed by three different nodes. The execution time for different tasks are denoted as t_1 , t_2 and t_3 , respectively; and n is the number of input datasets. The total computation time is reduced from $t_s = n * (t_1 + t_2 + t_3)$ to $t_p = (n - 2) * \max(t_1, t_2, t_3) + \max(t_1, t_2) + \max(t_2, t_3) + t_1 + t_3$. For example, assume $t_1 = t_2 = t_3 = t$, when n is large, the computation time is reduced from $t_s = 3nt$ to $t_p = (n + 2)t \approx nt$, thus a speedup of 3 is achieved.

Comparing with task parallelism, data parallelism distributes data across multiple parallel nodes as shown in Fig. 2.2(c). The same task is executed in each of the parallel node. Data parallelism is especially useful for point-by-point operations with large datasets that can be very efficiently parallelized when the number of parallel nodes is large.

2.4.2.2 Amdahl's Law^[81]

Usually not all the parts of a given problem can be parallelized, and the efficiency of improvement using parallel computing can be roughly estimated by Amdahl's Law [81]. Assume in a problem, the execution time of the computations that can and cannot be parallelized be t_p and t_u respectively; $p = t_p / (t_p + t_u)$ is the percentage of the parallelizable parts; and n is the amount of improvement by parallel computing. The total execution time is reduced from $t_p + t_u$ to $t_p / n + t_u$, and the overall speedup is,

$$s = \frac{1}{(1-p) + p/n} \quad (2.40)$$

Let us consider two extreme cases: When p is very large and approaching 1, $s \approx n$ is obtained, which means that speedup increases with the number of the nodes. This case is called embarrassingly parallel problem [57] that can be most efficiently parallelized. When p is very small and approaches 0, $s \approx 1$, which shows that the problem is sequential thus cannot be accelerated by parallel computing. It is worth mentioning that Amdahl's Law only addresses a theoretically predicted speedup. In practical cases, the extra time introduced by the parallel computing should also be taken into account.

2.4.2.3 Hardware Platforms

During the past few decades, the parallel platforms have also been developed with the development of semiconductor techniques. Based on Flynn's taxonomy [82, 83], hardware systems can be classified into four architectures: single-instruction-single-data (SISD), single-instruction-multiple-data (SIMD), multiple-instruction-single-data (MISD) and multiple-instruction-multiple-data (MIMD). SISD is equivalent to the sequential hardware architecture; SIMD is for data parallelism; MISD is rarely used; and MIMD is for task parallelism. It should be noted that although MIMD targets at task-parallel problems, data-parallel problems can also be accelerated in an MIMD hardware with same copy of program executed in each unit, though not efficiently. Many hardware platforms are hybrids of both SIMD and MIMD architectures.

➤ MIMD platforms

The common parallel hardware based on MIMD architecture include computer clusters, shared memory multiprocessors, general-purpose multicore processors, and pipeline processors. The grid computing [84] and cloud computing [85] are also common parallel hardware platforms but are not discussed below because they are for grand challenge problems [86] such as protein folding, financial modeling and climate modeling.

- A computer cluster [87] is a group of linked computers collaborating to solve a computationally expensive problem. The computers in a cluster are usually connected through a fast local link or network. The principle of a computer cluster is usually used to build a supercomputer.
- In shared memory processors (SMPs), the processors communicate through shared memory instead of link or network [87]. Compared with computer clusters, the communication time is greatly reduced and the system performance is improved accordingly. Like computer clusters, SMPs are also building blocks of supercomputers.
- A general-purpose multicore processor [88] integrates two or more independent cores into a single chip. From POWER4 by IBM [89] to i7 by Intel [90], multicores have been the primary factor to increase the performance of general-purpose computers. Prevalence of multicore processors in commodity PCs enables the parallel resources more easily available to the industry and scientific research. Kim and Bond [91] surveyed the state of the art scientific or programming software that had incorporated parallel computing libraries to simplify the complicated parallelizing techniques. To utilize the parallel units in a multicore CPU, one can implement the algorithm in multithreaded form [91], use SSE (streaming SIMD extensions) instructions [92], or directly use parallel toolboxes or libraries in many software development environments, I.E. MATLAB[®] parallel computing toolbox [59].
- A pipeline image processor [93] substantially converts a 2-D image consisting of a 2-D array of pixels into a 1-D stream of pixels then process them in a pipelined

form for simultaneous execution. At last, the processed pixels stream is transformed back into a 2-D image. It is a specialized hardware aiming at acceleration of the image processing problems. Now, the pipeline image processors are rarely seen for algorithm acceleration.

➤ SIMD platforms

Typical parallel hardware based on SIMD structure consists of vector processor, SIMD parallel processors and graphics processing unit (GPU).

- A vector processor [94] (array processor) is a CPU that can perform operations on multiple data elements simultaneously. Used from 1980s to 1990s in scientific computing area, vector processors were popular elements of most supercomputers. Nowadays, it is nearly obsolete and replaced by more efficient and inexpensive substitutes such as GPU. We did not find its application in the literatures of experimental mechanics and optical measurement.
- SIMD parallel processors [95] include multiple microprocessors arranged with SIMD architecture. A series of supercomputers CM-1, CM-2, CM-200 by Thinking Machines Corporation belong to the models of SIMD parallel processors [95].
- As the main processor on a commodity graphics card, GPU [96] uses thousands of stream processors [97] to increase the chip-level parallelism. During recent years, General-Purpose computation on GPUs (GPGPU) [60] is proposed to leverage the

powerful GPU capability to accelerate general purpose computations besides the graphics applications. Recently, computer unified device architecture (CUDA) [98] for NVIDIA GPUs facilitates the programming and enables GPUs being one of the popular parallel hardware. It should be noted that high computational capability of GPUs contributes to many models of top10 supercomputer in the world [99].

➤ Hybrid platforms

Many of the parallel hardware platforms combine both MIMD and SIMD processing units such as PC with multicore CPUs and many-core GPUs, Cell Broadband Engine (Cell BE), SGI Onyx computer and digital signal processor (DSP). In other hardware platforms such as Application Specific Integrated Circuit (ASIC) and FPGA, different parallelism hardware architectures can be designed by users.

- A PC can be a hybrid parallel system by simply equipped with a multicore CPU and many-core GPUs [100]. Data-parallel and time consuming computations can be offloaded to the GPUs, while the multicore CPU is in charge of handling the sequential or task-parallel portion of an algorithm. The codes for both CPU and GPU can be mixed in a program, while the underlying compiler of CUDA automatically extracts the instructions for GPU execution. By this very cost-effective way, huge speedup performance can be achieved.

- The Cell BE [101] was developed by Sony computer entertainment, Toshiba, and IBM. It combines a symmetric multi-threaded Power Processing Element (PPE) based on MIMD architecture and eight Synergistic Processing Elements (SPEs) based on SIMD architecture. Although Cell BE is one of the popular parallel computing platforms, we did not find its application in literatures of experimental mechanics and optical measurement so far.
- The SGI Onyx [102] is a heterogeneous computer with both MIMD and SIMD architecture. The computer can be equipped with one or multiple processors and a specialized graphics subsystem. During 1996 to 2003, it was a high-end product designed by Silicon Graphics Inc. (SGI) for graphics rendering and visualization. Now Onyx series are nearly obsolete.
- A DSP [57] is a fast specialized microprocessor, aiming at digital signal processing and embedded systems. With multiple functional units and VLIW (very long instruction word)/SIMD design, it has been widely applied in industries which require real-time performance, such as mobile phones, multimedia computers, video encoders and CD players [96]. Modern DSPs are also equipped with multicores.
- Exempted from the hardware restriction in general purpose CPUs, an ASIC [103] is usually tailored and provides the best performance for specific applications. It is widely used in consumer electronic products such as digital cameras, cell phones, and various control circuits in the industry [104]. However, due to long development cycle, ASICs are not economic and practical for lower production volumes.

- An FPGA [96] is used as the prototyping device for the ASIC design. With shorter time-to-market, it is regarded as the trade-off between the flexibility of programmability and the speed of an ASIC. In FPGA design, functions can be realized by mapping to the reconfigurable logic blocks and interconnects using hardware description languages. Many FPGAs can even be dynamically reconfigured to reduce energy and power consumption of a system [105, 106]. However, compared with other processors with fixed hardware architecture, the development cycle of FPGA is longer due to the across synthesis issue [107]. FPGAs can be configured with DSP units and intellectual property (IP) cores [108] to shorten the development cycle.

2.4.2.4 Pros and cons of parallel hardware platforms

In general, different platforms have their own pros and cons, and there is no universal solution to all the applications. To select a proper hardware platform for a specific application, several issues should be considered such as the parallel type of the problem, performance per cost, energy consumption, and usually the development cycle. Among thirteen platforms introduced above, pipeline image processors, vector processors, SIMD parallel processors and SGI Onyx computers are rarely seen nowadays; ASICs are more suitable for developing commercial products; the remaining platforms can be classified into the higher level platforms (computer clusters, SMPs, and PCs with a multicore CPU and GPUs) and the processor level platforms (general purpose multicore processors, GPUs, DSPs, Cell BEs and FPGAs) by which the former can be built. Among the higher

level platforms, computer clusters usually have limited speedup performance due to long communication time; SMPs have good performances but are at high cost; in contrast, PCs with a multicore CPU and GPUs provide good performance at a relatively low cost. The processor level platforms are compared in Table 1 and the latest model for each platform is provided. From Table 1, it can be seen that,

- Because of the SIMD architecture, GPUs are only suitable for data-parallel problems. Other platforms are suitable for all the task-parallel, pipeline-parallel, as well as data-parallel problems.
- Among them multicore CPUs and DSPs are mainly based on MIMD architecture, thus they have limited efficiency for the data-parallel problems. FPGAs and Cell BEs are universal and can be efficient for different parallelism types.
- A factor to evaluate the efficiency of a platform is the floating-point operations per second (flops). Performance per cost is defined as flops divided by cost. In Table 2.1, all flops are given as single precision (SP), since the SP is usually adequate for the computations in experimental mechanics and optical measurement. Although FPGAs focus on integer computations, the up-to-date FPGAs can be configured with DSP units to achieve very high throughput of the floating-point computations. Thus in Table 2.1, the flops for FPGAs are evaluated by the performance of its configurable DSP slices. As seen in Table 2.1, GPUs produce the best performance per cost. Although the performance per cost of FPGAs is relatively low, it is more flexible for various types of parallelism and can serve as the prototyping device for ASICs.

- The power consumption of FPGAs and DSPs are lower compared with that of other platforms. Therefore, they are usually excellent candidates for embedded system and SoC (system on a chip) [109].
- In the view of programmability, we can see that most platforms support C or C++ except for FPGAs. Although there are C-like programming languages for FPGAs such as HandleC [110], they are still based on hardware level, which users cannot rely on compilers to map the codes to the hardware efficiently and robustly. Consequently, the development cycle of FPGA is the longest [96].
- Although each platform has its pros and cons, a hybrid system can be constructed using the above processor level platforms. Different hardware can complement one another to give the optimum performance. For example, nearly all commercial PCs can be equipped with a multicore CPU and at least one graphics card; such hybrid parallel platform can be used for many parallel problems and is cost-effective.

Table 2.1 Comparison among processor-level hardware platform

	Multicore CPU (Intel i7-980XE) [90]	Cell BE (Playstation 3) [111]	DSP (TMS320C6678) [112]	FPGA (Virtex-6 SX475T) [113]	GPU (NVIDIA® GeForce® GTX580) [114]
Number of Cores	6	9	8	Configured	512
Architecture	MIMD	MIMD & SIMD	MIMD & SIMD	MIMD & SIMD	SIMD
Giga Flops	160 (Estimated)	204	160	129	1581
Cost (USD)	1000	600	400 (kit)	2000+ (kit)	600
Power consumption (Watts)	130	70-140	10	13+	244
Develop Languages & library support	C, C++, Java, MATLAB®, openMP[115],Pthreads[116]	C/C++ toolkit library, OpenCL [117]	DSP MCSDK [118], C,C++ library	VHDL, Verilog, HandleC, IP cores	C, C++ , CUDA Toolkit & SDK, OpenCL
Development cycle	Medium	Medium	Medium	Long	Medium

2.4.3 Applications in experimental mechanics and optical measurement: platforms based on MIMD

2.4.3.1 Computer clusters

Most works using computer clusters belong to the acceleration of holographic particle image velocimetry (HPIV), in which digital hologram reconstruction and particle tracking algorithms are time-consuming. Pu and Meng [119] adopted a distributed parallel computing cluster in the Gemini HPIV system. The cluster consisted of a master Pentium computer and eight slave Pentium computers, which were connected by a high-speed low latency network. Therefore, the computation time was reduced from 50 hours to a few hours. Satake et al. [120] published a similar work using a PC cluster to track micro particles in digital holographic particle tracking velocimetry (DHPTV) system. In their work, the hologram volume was divided into sections either along the time axis or in space domain, where the sections were distributed among the parallel PCs. They showed that the time division method for hologram reconstruction outperformed space division method and by using four PCs the speedup for the former was $3.3\times$ (times). Recently, Fugal et al. [121] extended the previous works, and proposed an auto tracking system for both small and large particles in digital in-line hologram. They adopted a robust algorithm for focus position detection by edge sharpness and accelerated it using a computer cluster. MATLAB[®] parallel computing toolbox was used. They claimed that their algorithm was

embarrassingly parallel and the speedup was linear with the number of nodes although the details of the performance were not mentioned in their paper.

Page et al. [122] made use of the distributed memory in a computer cluster to solve the problem of reconstructing large digital holograms. Targeting at not only a computer cluster but also multiple cores in a CPU, a parallel algorithm for Fresnel transform was proposed. By using 26 desktop PCs running at 2.0GHz with 1GB of memory, the total reconstruction time was reduced from more than 5 hours to 2.2 hours for a $2^{14} \times 2^{14}$ digital hologram. They could even be used to reconstruct large hologram with 4.3 Gigapixels. They also showed that there was no benefits using more than eight PCs. Ng et al. [123] proposed using single program multiple data of parallel computing for temporal fringe pattern analysis. A computer cluster with two workstations was used, each of which consisted of two Intel Xeon 3.2 GHz processors. Up to $1.6\times$ speedup was achieved for a simulated sequence of 512 temporal speckle interferometry fringe patterns with 512×480 pixels.

2.4.3.2 Shared memory multiprocessors (SMPs)

Satake et al. [124] developed a parallel algorithm for particle reconstruction and particle search in microdigital-holographic particle-tracking velocimetry using SGI Altix machine. Usually to process one second of data, the computing time for digital holograms with time-evolution measurement is tens of hours. By employing parallel computing and a SGI Altix machine consisting of 128 processors with 2.4 GHz, the computation time was

reduced from 439 seconds to 25 seconds for hologram reconstruction and searching algorithms. Although the shared memory system is scalable, saturation can be observed in their experiments because the overhead exceed the computation time. To overcome this bottleneck, they suggested their system for a larger computation scale with larger hologram and more depth for reconstruction direction [124].

2.4.3.3 General-purpose multicore processors

The parallel 3-D WT was implemented on multicore CPUs by Franco et al. [125]. By using three methods consisting of automatic parallelization by C compiler, semi-automatic parallelization using OpenMP, and explicit thread-level parallelism with Pthreads, they showed that the explicit parallel code using OpenMP and Pthreads outperformed the compiler parallel technique and achieved speedup of 6× to 12×.

In high-resolution real-time 3-D shape measurement proposed by Zhang and Huang [126], a dual-core computer was used, which was shown necessary for their real-time applications. After fringe images were captured by a high-speed CCD camera, one CPU core was in charge of the acquisition, while the other handled the reconstruction and display. Their software was multithreaded in a pipelined form. Frame rate of 40 fps for the resolution of 523×500 was obtained for real-time 3-D shape acquisition, reconstruction and display. In the measurement of nanoscale displacement, a real-time fringe fitting algorithm was proposed by Ionția et al. [127]. By parallelizing the

Levenberg-Marquardt algorithm, they achieved 30 fps using two cores in a computer with a quad-core processor.

2.4.3.4 Pipeline processors

Moore and Tyrer [128] used a pipeline image processor for phase stepping and successive fringe pattern analysis in 2-D strain measurement with ESPI. Coggrave and Huntley [55] proposed a high-speed surface profilometer using a spatial light modulator and a pipeline image processor. Phase shifting and temporal phase unwrapping were implemented on a Datacube VME system, which consisted of two 20 MHz pipeline processor boards. Their system achieved video rate to perform real-time 3-D surface analysis including real-time phase shifting and phase unwrapping. Later they applied the same system to speckle interferometry and achieved real-time visualization of deformation fields [129]. However, the phase unwrapping algorithm in their real-time 3-D system only consider accelerate simple phase unwrapping algorithm and cannot be applied to the fringe patterns with high level of noises and long discontinuities.

2.4.4 Applications in experimental mechanics and optical measurement: platform based on SIMD

2.4.4.1 SIMD parallel processors

In the work of Lucent [130], an algorithm for real-time interactive computation in computer generated holography (CGH) was proposed. By a pre-computed look-up table (LUT) for image plane hologram and a CM-2 supercomputer, a horizontal parallax only hologram from a 10,000-points object was generated in less than 1 second.

2.4.4.2 Graphics processing units (GPUs)

Most applications of GPUs in experimental mechanics and optical measurement fall into the massive data-parallel computation in the computer generated hologram (CGH). However, the large computational cost prohibits CGH from real-time applications. Compared with many hardware platforms that have been applied to the CGH, GPUs on graphics cards in a PC is a more cost-effective solution. Other applications of GPUs belong to digital holographic imaging (DHI), microscopy, and fringe pattern analysis.

Petz and Magnor [131] made the first attempt on using a GPU to accelerate the CGH computations. By using a NVIDIA Quadro GPU, the MIT Marl II holographic video system proposed by Bove Jr. et al. [132] could produce single-color horizontal parallax only holographic images at a frame rate of 2 fps. Plesniak et al. [133] employed PC-based graphics cards instead of expensive SGI Onyx system to compute reconfigurable image projection (RIP) holograms. They showed that 1.2 fps was achieved by using multiple GPUs. All of the above groups used graphics API and had to map the CGH algorithm to the graphics rendering process. The group from Chiba University [134] were also active in hardware acceleration. By using GPU in full-parallax CGH, the implementation was

based on a NVIDIA's GPU and Microsoft's High Level Shader Language (HLSL). The computation speed of a NVIDIA GeForce 6600 GPU was shown $47\times$ faster than that of a Pentium 4 CPU. Ahrenberg et al. [135] conducted similar work and used multiple GPUs to compute the hologram. Due to the limited power of the GPUs at that time, the CGH for large images was a tough task. Janda et al. [136] addressed a discrete computational procedure that yielded a planar discrete hologram from a given 3-D object. They also discussed using a cluster of computers with GPUs to accelerate the algorithm.

From 2008, the development of the CUDA for GPGPU has made GPUs popular for general purpose computing since the underlying hardware usage is more efficient and programming is easier. The following works reviewed for applications in CGH are based on GPUs with the CUDA. Kang et al. [137] adopted a GPU to compute a compensated phase-added stereogram and achieved 30 fps for objects with 15,000 points. Later they [138] proposed a multi-GPU approach for an accurate compensated phase-added stereogram (ACPAS). Experiments showed that the computation of the ACPAS on a multi-GPU platform was approximately $100\times$ faster than CPU implementation for a full color digital hologram. By using a novel split look-up table (LUT) algorithm and NVIDIA GTX285 GPUs, Pan et al. [139] obtained high speed, low memory usage and full parallax reconstruction. Chen and Wilkinson [140, 141] published two works based on GPUs in the CGH. They achieved full-parallax CGH supporting geometric occlusion as well as a basic lighting model of perfect diffuse reflection. Recently, the group from Chiba University [142] used an LUT and a new wavefront-recording plane (WRP) for rapid computations in the Fresnel CGH, and achieved 10 fps for objects with 30,000 points. The group [143, 144] also published another two papers on real-time color CGH.

Due to the high computational power of modern GPUs, a 3-D object with 1,000 points per color could be animated at a virtually video rate. Using a GPU for optimization of holograms via simulated annealing in the CGH was investigated by Carpenter and Wilkinson [145]. The GPU implementation was also compared with a parallelized counterpart in a mainstream quad-core PC, in which the former was $25\times$ faster than the latter. It was noted that recently the Chiba group [146] explored using an AMD GPU and OpenCL in CGH for the first time. This work can be seen as an alternative and competing solution of the previous works using the CUDA.

For other applications, Reicherter et al. [147] used a GPU for adaptive force measurement. To control the holographically generated traps, a fast hologram computation algorithm was implemented on a GPU and an average performance of 13.6 Gigafllops was achieved. Haist et al. [148] adopted a GPU to reconstruct a hologram for holographic tweezers in the object plane of a microscope. They programmed the early GPUs using Cg and OpenGL and achieved up to $17\times$ speedup than using a Pentium 4 CPU. The Chiba group [149] applied a GPU for digital holographic microscopy (DHM) and obtained 24 fps for 512×512 holograms. A new extended focused images (EFI) reconstruction method was proposed by Wang et al. [150] using FFT pruning with frequency shift to simplify the Fresnel transform method, coordinate transformation, and subpixel resampling. Due to the inherent parallelism in their algorithm, GPU and CUDA were adopted and 300-500 times speedup could be obtained comparing with sequential codes. In the work of Zhang et al. [151], a GPU was used to assist coordinates computation and rendering in high-resolution 3-D shape measurement using fringe projection, in which they realized real-time measurement and display at 30 fps for frames with the size of 532×500 . High-resolution

laser speckle imaging for fast blood flow visualization was shown by Liu et al. [152] using a GPU. They implemented Newton iteration in GPU for laser speckle contrast analysis and achieved video frame rate for 1920×1440 images. Recently a parallel, real-time algorithm was proposed by Espinosa-Romero and Legarda-Saenz [153] for the estimation of the dynamic phase map of a vibrating object by using a Fourier-based quadrature transform and temporal PU technique. They achieved real-time measurement and visualization with 1600 fps utilizing a GPU and CUDA.

2.4.5 Applications in experimental mechanics and optical measurement: platforms based on hybrid architectures

2.4.5.1 PC with a multicore CPU and GPUs

In the work of Cheong et al. [154], they proposed a system based on a multicore CPU and GPUs for in-line holographic particle tracking. Through a combination of hardware acceleration and software optimization, particle tracking and analysis could be realized in near real time.

2.4.5.2 SGI Onyx computers

During 1990's, many works adopted SGI Onyx computer for real-time computation and visualization of the CGH. Lucente and Galyean [155] first introduced an Onyx computer to compute holographic patterns. By using Onyx with a RealityEngine2 and another similar SGI computer, they could calculate a 6-megabyte holographic pattern in less than 2 seconds. In 1999, to accelerate full-parallax synthetic holograms, Ritter et al. [156] adopted Silicon Graphics Onyx2 computer which was the second generation of the original Onyx system including two R10000 (195 MHz) processors and Infinite Reality graphics hardware. By using this computer, good speed performance was gained but far from real-time applications. Halle and Kropp [157] also proposed a work using computer graphics workstations in fast full parallax spatial rendering. They mapped the algorithms of full parallax holographic stereogram into rendering API using OpenGL graphics library and implemented them in two Silicon Graphics computers, 150MHz R4400 Indigo2 workstation and a two processor 150MHz R4400 Onyx RealityEngine2. They reduced the time of rendering the stereogram to 1 second for a moderately sized scene.

2.4.5.3 Digital signal processors (DSPs)

One of the applications using DSPs is to control or accelerate the processes in the fringe pattern analysis. Burton et al. [158] reported carrier frequency shifting in Fourier transform fringe analysis for surface shape measurement. A DSP in an accelerator card was used for fringe analysis and 4 second was achieved for a 256×256 image. Doval et al. [159] obtained real-time contrast enhancement and phase control of fringes using a DSP in stroboscopic additive fiber optic TV-holography for vibration analysis. Lee et al. [160]

also utilized a DSP to control a phase shifter, which can be used in phase shifting technique for optical metrology.

Another application using DSP is for embedded CMOS cameras. Aguan et al. [161] adopted a CMOS camera integrated with a DSP in their heterodyne speckle interferometer for velocity profile reconstruction. Single pixel carrier based demodulation was implemented in the DSP. The system was proved as a low-cost alternative to expensive and complex devices producing quasi-instantaneous full-field velocity measurement. Later, they [162] adopted the same CMOS-DSP system along with holographic optical elements for real-time vibration detection. The detection frequency was able to achieve up to 100Hz.

Due to the fact that vibration could cause errors in surface metrology, Zhao and Burge [163] introduced a vibration-compensated interferometer using a DSP, a phase shifter, and a high-speed phase control from an electro-optic modulator. A closed-servo system was used for measuring the phase while a DSP was in charge of sending control commands as well as calculating the phase. By using a DSP to accelerate the least squares fitting algorithm, Zumberge et al. [164] realized processing quadrature fringes in real time. Park et al. [165] used a robust discrete time Kalman filter for real-time precision displacement measurement and implemented it in a DSP. Huang and Yeh [166] adopted a DSP to enhance the dynamic measurement and isolated various interferences in the optical measurement system for error motions of small shafts. A DSP was also used in live electro-optic imaging in the work of Sasagawa et al. [167]. The real-time detection of

magnitude and phase images was obtained for instantaneous visualization of radio frequency electric fields.

2.4.5.4 Application specific integrated circuits (ASICs)

Along with the work using ASIC, the works using electrical or electronic circuits during 1990's were also reviewed in this subsection. Nakadate and Isshiki [53] proposed 2-D and 3-D LUTs in digital memories and phase unwrapping circuits for real-time fringe pattern processing. A control circuit was reported by Racca and Dewey [168] in solid-state shutters for high speed time-resolved holographic interferometer. Their system achieved successful interferometric recordings of the reflection of a shock wave off a model at a frame rate of up to 10 kHz. In the paper of Yue-Guang et al. [169], hardware was designed for phase shifting in holographic interferometry. By an LUT and a logic circuit, calculation time for a phase with 512×512 pixels was 40 millisecond. A fringe analyzer [54] was introduced by Kato et al. for real-time phase distribution analysis from three phase-shifted moiré patterns, which were generated simultaneously by electronic multiplication with computer-generated reference gratings and low-pass filtering. The hardware adopted in the fringe analyzer included a phase-computation circuit and a phase-unwrapping processor. Although real-time phase retrieval was achieved, only simple PU algorithm was implemented and the system was restricted to the unwrapping of only full-field phase maps with no invalid areas or noises inside them.

2.4.5.5 Field Programmable Gate Arrays (FPGAs)

The FPGAs were extensively applied to accelerations in optical measurement by the group from Chiba University [170-177]. From HORN-2 to HORN-6, they had designed many generations of special-purpose computers based on FPGAs. Four large-scale FPGAs were used at 166MHz on the recently developed HORN-5 board and several HORN-5 boards could be connected into a PC to form an FPGA cluster by Peripheral component interconnect (PCI) bus. The HORN-5 board was used for computations in CGH [172]. A PC with four HORN-5 boards could calculate a hologram from a 3-D object consisting of 10,000 points at 30 fps, which was 1,000× faster than a Pentium 4 CPU. In another application, a HORN-5 board was used for fast Fourier transform (FFT) in DHPTV system. To reconstruct an image from a 256×256 hologram, the HORN-5 system obtained 38.1× speedup than a Pentium 4 CPU with 3.2GHz and 2G memory [173]. For a hologram with size 1024×1024, the HORN-5 board was able to produce 0.3 fps, which is 7× faster than that of a Pentium 4 CPU [174]. Lately, they improved their system by using a PC cluster with several HORN-5 boards. The cluster consisted of one host PC and four PCs with HORN-5 boards in each of them and the PCs were connected by a gigabit Ethernet. By using forty-eight FPGA chips in the cluster, the computation speed of the cluster was 110× faster than that of a PC [175]. The improvement of HORN-6 system over the HORN-5 board was mainly an extra external memory module [176]. Thus, the amount of memory that could be used was about 350× larger than that of the HORN-5. By using an improved HORN-6 cluster system, the reconstruction of hologram could achieve 10 fps for objects with 100,000 points and 2× faster than using a HORN-5

cluster with the same number of FPGA chips. Besides developing the hardware platform, they [177] put efforts on the parallel fix-point algorithm targeting at specialized processor such as FPGAs, DSPs and ASICs. An efficient method for computation in the CGH was given using addition instead of multiplication. Along with an LUT method, they showed that their method was $24\times$ faster than a direct computation. Furthermore, a one-unit system was proposed by the group for 3-D movie or 3-D TV. Targeting at embedded applications, one or two connected boards were developed consisting of FPGAs as processors and a high-resolution, reflective mode, liquid-crystal display panel as a spatial light modulator [178, 179]. By using this one-unit system, they obtained real-time CGH at 30 fps for a 10,000-points object.

In the work by Lenart et al. [180], a Xilinx Virtex-1000E FPGA was integrated into a microscope based on the Digital Holography (DH) to reconstruct images captured with a sensor device in which the FPGA targeted at computing a 2048×2048 2-D FFT. By employing a fast transpose unit and an interleaved memory storage scheme, the FFT calculation was $3\times$ faster than the traditional column/row approach. Thus real-time hologram reconstruction could be achieved with 20 fps. Veitch et al. [181] were also involved in using an FPGA for holographic reconstruction and designed a pipelined Fresnel-Kirchhoff transform. With the FPGA running at up to 75MHz, each reconstruction took 25 microseconds for a hologram with 1024×1024 pixels. Hanak et al. [182] addressed an acceleration algorithm for reduced occlusion method in optical field synthesis of CGH. The algorithm was optimized by a fixed-point method and was implemented in an FPGA. They demonstrated that a board with four FPGAs were able to calculate 7,500-point sources per millisecond per subpart of the hologram. An FPGA was

also integrated into a controller board for high-speed spatial light modulators by Höfling and Ahl [183]. The board was adopted by Ri et al. [184] for single-shot 3-D shape measurement by fringe projection. Recently a lifting-based discrete WT on an FPGA was introduced by Gao and Xiong [185], revealing the potential of FPGAs in fringe pattern analysis. By employing a CMOS sensor and an FPGA, Fujigaki et al. [186] developed a sampling moiré camera to analyze 2-D grating images in real time. Other components consisted of the memory modules, a USB interface and a power unit. Experiments were implemented to measure a cantilever as a real-time application. Buckley [187] proposed a real-time error diffusion algorithm to enhance the SNR in a holographic projection system. The algorithm was implemented in an FPGA and 16fps was achieved for 512×512 binary phase holograms.

2.4.6 Discussions and conclusions

In Section 2.4.2, various hardware platforms for the acceleration of different problems were introduced. In Sections 2.4.3-2.4.5, their applications in experimental mechanics and optical measurement were reviewed to show different characteristics of them. From these sections, the advantages of using parallel hardware platforms to accelerate the algorithms are obvious. As for the selection of hardware platforms, Section 2.4.2 could be used as an initial guideline. To make a wise selection, we could gain a deeper knowledge on various applications of hardware platforms and better understanding of the project to be implemented.

Analysis of the distribution of the reviewed papers is interesting. In Fig. 2.3(a), we can see that for the reviewed 69 papers during the past decades, about 31% use GPUs, 25% use FPGAs, and 44% use others. For the literatures from 2008, about 43% use GPUs, 31% use FPGAs, and 26% use others as shown in Fig. 2.3(b). It can be seen from the above statistics that many problems accelerated in experimental mechanics and optical measurement are data parallel, for which GPUs and FPGAs are excellent candidates. Furthermore, the CUDA boosts the works using GPUs in the recent four years because the programming is easier and the hardware usage is more efficient. Using FPGAs requires accumulation of hardware expertise and the development cost is higher compared with GPUs, as discussed by the Chiba group [146]. For data-parallel problems, GPUs are more cost-effective especially in laboratory environment.

Figure 2.3 Distribution of the reviewed literatures (a) Total (b) The literature from 2008. Reprint from Ref. [77] with permission from Elsevier.

Tomography is categorized into an optical measurement technique and has been widely used in many areas. The application of parallel hardware in tomography has been reviewed by Ni et al. in [188] and thus not covered in this Chapter. Adaptive optics is also a closely related area where parallel computing has been extensively used. To keep the chapter concise, the review of adaptive optics is not included.

2.5 Conclusions

In this chapter, we first reviewed the popular PU algorithms including the path following PU algorithms, the minimum L^P norm algorithms, the Bayesian/ regularization PU algorithms and the PU algorithms based on parametric polynomial models. For the phase with noises and short discontinuities, most PU algorithms can obtain satisfied phase unwrapping results. For the phase with long discontinuities, 1) it is seen that when high-reliable quality maps are not available, nearly all path following PU algorithms and minimum L^P norm algorithms may introduce phase errors except for the WFF-QG algorithm; 2) Only the latest Bayesian/ regularization PU algorithms [17-22] can obtain good results; 3) Among the PU algorithms based on parametric polynomial models, the WFR algorithms and the WR algorithm can usually achieve satisfied results for this type of phase. For the phase with alias, most Bayesian/ regularization PU algorithms [15-22] can get successful unwrapping results while other categories of PU algorithms cannot.

It is seen from above discussion that WFT-based QG algorithms and latest Bayesian/ regularization PU algorithms can achieve reliable results for phase with long

discontinuities. Compared with the latest Bayesian/ regularization PU algorithms [17-22] whose computation time depend on both the size and the complexity (Higher complexity means higher level of noise or under sampling and more areas with true phase discontinuities) of the fringe pattern, the computation time of WFT-based QG only depends on the size of fringe patterns once the parameters are set. Thus the WFT-based QG PU algorithms are more suitable for high-speed phase retrieval.

Secondly, the principles for the WFT, CRB, the MLE and preliminaries of first order perturbation were introduced as a background of statistical analysis which will be addressed in the later Chapters. At last, the basic principles of parallel computing and different architectures of the parallel platforms were briefly introduced. The pros and cons of current popular hardware platforms were introduced and the works using parallel platforms to boost the system performance were reviewed. From our review, we can see that for the WFT based algorithms which are typical data-parallel problems, GPU is a most promising candidate.

It should be noted that the review of hardware acceleration is more detailed than other parts. This is because other parts of this Chapter provide a background to serve this dissertation, while the hardware acceleration not only serves this dissertation but also provides an easy entrance for mechanical and optical engineers who are less familiar with hardware acceleration. The thorough review of the hardware acceleration is one contribution of this thesis.

Chapter 3 Phase compensation for the WFR algorithm

3.1 Introduction

As discussed in Chapter 1, the WFR algorithm is an effective algorithm for fringe pattern analysis. It can suppress the noises as a preprocessing step for phase retrieval [41]; it is shown more accurate than the FT, WFF and WR algorithm in phase extraction from a single fringe pattern for a continuous phase [49]; and it can be used to extract the frequencies from the wrapped phase map [41]. However, a small bias is produced using the WFR algorithm to reconstruct the phase, which reduces the accuracy of the WFR algorithm for phase reconstruction. To increase the accuracy of the WFR algorithm, we propose two methods to compensate for this phase bias. The first one is using linear equations and two frequency points on the WFT spectrum, while the second method is based on the least squares fitting principle and convolution method to correct the phase.

Due to the lack of ground truth in experimental fringe patterns, we first provide the processing results of the proposed methods on simulated fringe patterns. The proposed WFR algorithm with phase compensation methods (the WFRC algorithm) is tested on fringe patterns with simulated additive Gaussian noise or speckle noise [189]. Then, for

simulated fringe patterns with long discontinuities, the WFRC algorithm will be shown to produce a better quality map than that of the WFF algorithm for the quality-guided PU algorithm. Secondly, the WFRC algorithm is tested on an experimental wrapped phase map in fringe projection profilometry to show its accuracy. By using the results of several phase unwrapping algorithms for reference, it will be shown that the proposed methods are effective to compensate for the phase bias even in experimental fringe patterns.

3.2 2-D windowed Fourier ridges algorithm

Given a phase model without noise by Eqs. (2.28) and (2.29), a forward 2-D windowed Fourier transform is applied to the EPF without noise.

$$\begin{aligned} Sf(u, v; \xi, \eta) &= Sf_0(u, v; \xi, \eta) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) g(u - x, v - y) \exp(-j\xi x - j\eta y) dx dy, \end{aligned} \quad (3.1)$$

where $Sf(u, v; \xi, \eta)$ is the windowed Fourier spectrum of a input fringe pattern $f(x, y)$ and $g(u - x, v - y)$ is a real and symmetric window function to localize the Fourier integral within the neighborhood of a point at (u, v) . Usually a Gaussian window is used,

$$g(x - u, y - v) = \frac{1}{\sqrt{\pi\sigma_x\sigma_y}} \exp\left[-\frac{(x - u)^2}{2\sigma_x^2} - \frac{(y - v)^2}{2\sigma_y^2}\right], \quad (3.2)$$

with σ_x and σ_y as its kernel size.

Through some derivations [50], the following results can be obtained,

$$Sf(u, v; \xi, \eta) = A(u, v; \xi, \eta) \exp[j\Phi(u, v; \xi, \eta)], \quad (3.3)$$

where

$$A(u, v; \xi, \eta) = b \left[\frac{16\pi^2 \sigma_x^2 \sigma_y^2}{(1 + \sigma_x^4 c_{20}^2)(1 + \sigma_y^4 c_{02}^2)} \right]^{1/4} \exp \left\{ -\frac{\sigma_x^2 [\xi - \omega_x(u, v)]^2}{2(1 + \sigma_x^4 c_{20}^2)} - \frac{\sigma_y^2 [\eta - \omega_y(u, v)]^2}{2(1 + \sigma_y^4 c_{02}^2)} \right\}, \quad (3.4)$$

$$\begin{aligned} \Phi(u, v, \xi, \eta) = \varphi(u, v) - \xi u - \eta v - \frac{\sigma_x^4 c_{20} [\xi - \omega_x(u, v)]^2}{2(1 + \sigma_x^4 c_{20}^2)} - \frac{\sigma_y^4 c_{02} [\eta - \omega_y(u, v)]^2}{2(1 + \sigma_y^4 c_{02}^2)} \\ + 0.5 \arctan(\sigma_x^2 c_{20}) + 0.5 \arctan(\sigma_y^2 c_{02}) \end{aligned} \quad (3.5)$$

It is obvious that $|Sf(u, v; \xi, \eta)|$ is maximized when $\xi = \omega_x(u, v)$ and $\eta = \omega_y(u, v)$. By substituting $\xi = \omega_x(u, v)$ and $\eta = \omega_y(u, v)$ into Eq.(3.3), we can retrieve the phase of the input EPF. Thus the WFR algorithm can be established as follows,

$$[\hat{\omega}_x(u, v), \hat{\omega}_y(u, v)] = \arg \max_{\xi, \eta} |Sf(u, v; \xi, \eta)| = \arg \max_{\xi, \eta} |Sf(u, v; \xi, \eta) \exp(j\xi u + j\eta v)|, \quad (3.6)$$

$$\Phi[u, v, \hat{\omega}_x(u, v), \hat{\omega}_y(u, v)] = \text{angle} \left\{ Sf[u, v, \hat{\omega}_x(u, v), \hat{\omega}_y(u, v)] \exp[j\hat{\omega}_x(u, v)u + j\hat{\omega}_y(u, v)v] \right\}. \quad (3.7)$$

If we expand the left side of Eq. (3.7) using Eq. (3.5), we can see that the WFR algorithm produce a small phase bias $0.5 \arctan(\sigma_x^2 c_{20}) + 0.5 \arctan(\sigma_y^2 c_{02})$. To reconstruct the true phase, this bias should be subtracted from Eq. (3.7) as follows,

$$\begin{aligned} \hat{\phi}(u, v) = \text{angle} \left\{ Sf[u, v; \hat{\omega}_x(u, v), \hat{\omega}_y(u, v)] \exp[j\hat{\omega}_x(u, v)u + j\hat{\omega}_y(u, v)v] \right\} \\ - 0.5 \arctan(\sigma_x^2 \hat{c}_{20}) - 0.5 \arctan(\sigma_y^2 \hat{c}_{02}) \end{aligned} \quad (3.8)$$

where $\hat{\omega}_x(u, v)$, $\hat{\omega}_y(u, v)$, $\hat{\phi}(u, v)$, $\hat{c}_{20}$ and $\hat{c}_{02}$ are estimates of $\omega_x(u, v)$, $\omega_y(u, v)$, $\varphi(u, v)$, c_{20} and c_{02} , respectively.

Equation (3.8) is called phase compensation throughout the thesis. To compensate the phase, the second order derivatives of the phase $\hat{c}_{20}$ and $\hat{c}_{02}$ need to be estimated first, then $\hat{\varphi}(u, v)$ can be recovered by simple subtraction.

3.3 $\hat{c}_{20}$ and $\hat{c}_{02}$ estimates by linear equations

Since the frequencies along with x and y axis can be represented by the first order derivatives of the phase, the frequencies represented by Eqs. (2.30) and (2.31) can be used to estimate $\hat{c}_{20}$ and $\hat{c}_{02}$. As shown in Eq. (2.30), if we know the two frequency points $\omega_x(u_1, v_1)$ and $\omega_x(u_2, v_1)$, $\hat{c}_{20}$ can be solved from simple linear equations.

$$\hat{c}_{20} = [\hat{\omega}_x(u_2, v_1) - \hat{\omega}_x(u_1, v_1)] / (u_2 - u_1). \quad (3.9)$$

Similarly, $\hat{c}_{02}$ can be estimated by the following equation,

$$\hat{c}_{02} = [\hat{\omega}_y(u_1, v_2) - \hat{\omega}_y(u_1, v_1)] / (v_2 - v_1). \quad (3.10)$$

It should be noted that since the assumption is made as locally quadratic, the distance between two frequency points should not be large. In practice, the distances are selected as $s_x = \lfloor 2.5\sigma_x \rfloor$ and $s_y = \lfloor 2.5\sigma_y \rfloor$, where $\lfloor \cdot \rfloor$ indicate the floor operation.

3.4 $\hat{c}_{20}$ and $\hat{c}_{02}$ estimates by least squares fitting

Another method to compensate the phase is using least squares fitting to estimate $\hat{c}_{20}$ and $\hat{c}_{02}$ for phase compensation. In the least squares fitting perspective, we try to minimize

the differences between estimated local frequency $\omega_x(x, y)$ and expected local frequency $c_{20}x + c_{10}$ along x axis as follows,

$$R^2 = \sum_{i=1}^n [\omega_x(x_i, y) - (c_{20}x_i + c_{10})]^2, \quad (3.11)$$

where $\omega_x(x_i, y)$ are input frequency samples along x direction and x_i are their indexes.

The partial derivatives with respect to c_{20} and c_{10} gives,

$$\frac{\partial(R^2)}{\partial c_{20}} = -2 \sum_{i=1}^n [\omega_x(x_i, y) - (c_{20}x_i + c_{10})]x_i = 0, \quad (3.12)$$

$$\frac{\partial(R^2)}{\partial c_{10}} = -2 \sum_{i=1}^n [\omega_x(x_i, y) - (c_{20}x_i + c_{10})] = 0. \quad (3.13)$$

This leads to the equations in matrix form as,

$$\begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \begin{bmatrix} c_{10} \\ c_{20} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n \omega_x(x_i, y) \\ \sum_{i=1}^n \omega_x(x_i, y)x_i \end{bmatrix}. \quad (3.14)$$

The estimates of $\hat{c}_{20}$ and $\hat{c}_{10}$ can be derived by inversing Eq.(3.14).

$$\begin{bmatrix} \hat{c}_{10} \\ \hat{c}_{20} \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^n \omega_x(x_i, y) \\ \sum_{i=1}^n \omega_x(x_i, y)x_i \end{bmatrix}. \quad (3.15)$$

Consequently, $\hat{c}_{20}$ can be calculated according to Eq. (3.15) as,

$$\hat{c}_{20} = \frac{\sum_{i=1}^n \omega_x(x_i, y)x_i - \frac{1}{n} \sum_{i=1}^n \omega_x(x_i, y) \sum_{i=1}^n x_i}{\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2} = \frac{\sum_{i=1}^n (x_i - \bar{x})(\omega_x(x_i, y) - \bar{\omega}_x(x, y))}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad (3.16)$$

where $\bar{x}$ and $\bar{\omega}_x(x, y)$ are the average value of the x_i and $\omega_x(x_i, y)$ respectively. The numerator of Eq. (3.16) can be easily implemented by convolution and the denominator is the variance of x_i . Similarly, the $\hat{c}_{02}$ estimate can be derived from the following equation,

$$\hat{c}_{02} = \frac{\sum_{i=1}^n \omega_y(x, y_i) y_i - \frac{1}{n} \sum_{i=1}^n \omega_y(x, y_i) \sum_{i=1}^n y_i}{\sum_{i=1}^n y_i^2 - \frac{1}{n} \left(\sum_{i=1}^n y_i \right)^2} = \frac{\sum_{i=1}^n (y_i - \bar{y}) (\omega_y(x, y_i) - \bar{\omega}_y(x, y))}{\sum_{i=1}^n (y_i - \bar{y})^2}, \quad (3.17)$$

where $\omega_y(x, y_i)$ are input frequency samples along y direction and y_i are their indexes, $\bar{\omega}_y(x, y)$ and $\bar{y}$ are the mean value of $\omega_y(x, y_i)$ and y_i respectively.

3.5 Verifications

In this section, the WFR algorithm with the proposed phase compensation methods are first tested on two simulated wrapped phase with additive white Gaussian noise. The SNR for the additive Gaussian noise is 4dB. Figure 3.1(a) shows the first simulated wrapped phase with parameters $b=1$, $c_{00}=1$, $c_{10}=0.05$, $c_{20}=1 \times 10^{-3}$, $c_{01}=0.03$ and $c_{02}=2 \times 10^{-3}$. Then another simulated cubic wrapped phase shown in Fig. 3.2(a) is used to test the effectiveness of the WFRC algorithm on higher order polynomials. Its parameters are as follows, $b=1$, $c_{00}=1$, $c_{10}=0.05$, $c_{20}=5 \times 10^{-5}$, $c_{30}=2 \times 10^{-5}$, $c_{01}=0.03$, $c_{02}=5 \times 10^{-4}$ and $c_{03}=2 \times 10^{-5}$, where c_{30} and c_{03} are third order coefficients of the simulated cubic polynomial. To simulate the experimental fringe patterns according to Ref [190], the WFRC algorithm are also tested on the same quadratic phase as Fig. 3.1(a) but with severe speckle noise shown in Fig. 3.3(a). Then, the proposed WFRC algorithm is used to

process a simulated wrapped phase map with sharp edges before the quality-guided PU algorithm. This case will show the WFRC algorithm in improving the success of the quality guided PU algorithm. . Finally, to show the accuracy of the WFRC algorithm in a experimental interferometric pattern, we apply the WFRC algorithm on a fringe pattern obtained from fringe projection profilometry (FPP) [191, 192]. For the above verifications, the WFR algorithm and the WFF algorithm are also employed for comparison.

Figure 3.1 Testing on an simulated wrapped phase map with quadratic phase and Gaussian noise (a)

The simulated EPF with additive Gaussian noise 4db (b) The reconstructed phase by the WFRC algorithm, Phase error in radians of (c) the WFR algorithm (d) the WFR algorithm with linear equations for phase compensation(e) the WFR algorithm with least squares fitting for phase compensation (f) the WFF algorithm

The parameters for the WFR algorithm are $\sigma_x = \sigma_y = 10$ and the frequencies for both x and y axis are searched from -1 to 1 with a step of 0.001 . The parameters for the WFF algorithm are the same as that of the WFR and 3 is selected as the threshold. In implementation, the truncated window size for the WFR and WFF algorithms are $3\sigma_x + 1$ and $2\sigma_x + 1$, respectively. Due to the boundary effect [41] caused by convolution, 30 rows and 30 columns on all the four boundaries are not considered.

3.5.1 For simulated wrapped phase map with Gaussian noise

Although the simulated fringe pattern is very noisy, the proposed phase compensation methods are robust. Figure 3.1(b) shows the unwrapped phase by the WFRC algorithm. Figures 3.1(c-f) show the phase error in radians by the WFR algorithm, WFR algorithm with linear equations method for phase compensation, WFR algorithm with least squares fitting method for phase compensation and the WFF algorithm respectively. It can be shown from Fig. 3.1(c) that the WFR algorithm produces a small phase bias with about -0.15 rad. By using the phase compensation methods, this bias can be successfully suppressed, which is shown in Figs. 3.1(d) and (e). The phase errors produced by two compensation methods are in the same order of magnitude. However, the least squares fitting for phase compensation is more accurate than the linear equations method with slightly lower MSEs. It is also shown from Figs 3.1(d-f) that the WFRC algorithm

outperforms the WFF algorithm for the quadratic phase. The MSEs for the WFR algorithm, the WFR with linear equations method for phase compensation, the WFR with least squares fitting for phase compensation and the WFF algorithm are $2.28 \times 10^{-2} \text{ rad}^2$, $2.40 \times 10^{-4} \text{ rad}^2$, $2.01 \times 10^{-4} \text{ rad}^2$, and $1.0 \times 10^{-3} \text{ rad}^2$, respectively.

Figure 3.2 Testing on an simulated wrapped phase map with cubic phase and Gaussian noise(a) The simulated EPF with additive Gaussian noise 4db (b) The reconstructed phase by the WFRC algorithm, Phase error in radians of (c) the WFR algorithm (d) the WFR algorithm with linear equations for phase compensation (e) the WFR algorithm with least squares fitting for phase compensation (f) the WFF algorithm

For the simulated phase with cubic polynomial shown in Fig. 3.2(b), its wrapped phase map is shown in Fig. 3.2(a). Then the phase error (also in radians) by the WFR algorithm only, by the WFR algorithm with the proposed two phase compensation methods, and by the WFF algorithm are shown in Figs. 3.2(c-f) respectively. It can be shown that although the proposed phase compensation methods are based on quadratic polynomial model, they even works in higher order polynomials. The MSEs for the WFR algorithm without phase compensation, the WFR with linear equations for phase compensation, the WFR with least squares fitting for phase compensation, and the WFF algorithm are $2.45 \times 10^{-2} \text{ rad}^2$, $2.76 \times 10^{-4} \text{ rad}^2$, $2.99 \times 10^{-4} \text{ rad}^2$, and $2.8 \times 10^{-3} \text{ rad}^2$, respectively.

3.5.2 For simulated wrapped phase map with speckle noise

Figure 3.3(b) shows the reconstructed phase by the WFRC algorithm for the simulated wrapped phase with speckle noise which is shown in Fig. 3.3(a). Figures 3.3(c-f) show the phase error in radians by the WFR algorithm, WFR algorithm with linear equations method for phase compensation, WFR algorithm with least squares fitting method for phase compensation and the WFF algorithm respectively. Similar results are observed from Fig. 3.3(c-f) as the Fig. 3.1. The MSEs for the WFR algorithm, the WFR with linear equations method for phase compensation, the WFR with least squares fitting for phase compensation and the WFF algorithm are $2.32 \times 10^{-2} \text{ rad}^2$, $4.87 \times 10^{-4} \text{ rad}^2$, $4.35 \times 10^{-4} \text{ rad}^2$, and $1.7 \times 10^{-3} \text{ rad}^2$, respectively.

Figure 3.3 Testing on an simulated wrapped phase map with quadratic phase and speckle noise (a) The simulated EPF with speckle noise (b) The reconstructed phase by the WFR algorithm , Phase error in radians of (c) the WFR algorithm (d) the WFR algorithm with linear equations for phase compensation (e) the WFR algorithm with least squares fitting for phase compensation (f) the WFF algorithm

3.5.3 For a simulated wrapped phase map with long discontinuities

Figure 3.4 Testing on an simulated wrapped phase map with 3dB noise (a) A simulated fringe pattern with long edges (b) The quality map by the WFR algorithm and least squares fitting for phase compensation (c) The quality map by the WFF algorithm (d) The unwrapped phase by the WFR-QG (e) 3-D surface in radians of the unwrapped phase by the WFR-QG (f) 3-D surface in radians of the unwrapped phase by the WFF-QG

The proposed two phase compensation methods are also tested on a simulated wrapped phase map with long discontinuities, which is difficult to unwrap [19]. Figure 3.4(a) shows the simulated wrapped phase map with severe noises as well as a long discontinuity. To show the effectiveness of the proposed WFR-QG algorithm, we also provide the results by existing WFF-QG algorithm for comparison. Figures 3.4(b) and (c) show the amplitude which can be used as quality map produced by the WFR algorithm

and WFF algorithm in the WFRC-QG and WFF-QG algorithm, respectively. Since the accuracy of the proposed two phase compensation methods are in the same order of magnitude, we only show the phase compensation result by linear equations method to make the thesis concise.

It can be shown that the WFRC-QG algorithm can effectively indentify the long discontinuities for this simulated fringe pattern which the WFF-QG algorithm cannot differentiate from the fringes. Figure 3.4(d) shows the successfully unwrapped phase by the WFRC-QG algorithm. The 3-D phase surfaces in radians for both the WFRC-QG and the WFF-QG results are given in Figs. 3.4(e) and (f) respectively. Obviously, the WFRC-QG algorithm succeeds to unwrap this wrapped phase map except for small areas with discontinuities. However, the WFF-QG algorithm totally fails in phase unwrapping in this case. That is mainly because the WFRC algorithm is more sensitive to the edges than the WFF algorithm.

3.5.4 For an experimental wrapped phase map obtained from fringe projection profilometry

Figure 3.5 Testing on a experimental wrapped phase map from FPP (a) The wrapped phase map of a bump (b) 3D reconstructed phase by the multi-frequency method (c) 3D reconstructed phase in radians by the WFF-QG algorithm (d) 3D reconstructed phase in radians by the WFRC-QG algorithm (e) A comparison plot of unwrapped results for above three methods with line $y=130$ (f) An enlarged view of (e)

In the last experiment, we test the proposed WFRC algorithm on a wrapped phase map of a bump obtained from fringe projection profilometry. Due to the lack of ground truth, we also use the multi-frequency phase unwrapping method [193] and WFF-QG algorithm as references. Figure 3.5 (a) shows the wrapped phase map of the bump sample after phase shifting. Figures (b-d) shows the phase unwrapping result by the multi-frequency method, WFF-QG algorithm and WFRC-QG algorithm, respectively. It can be seen from Fig.

3.5(b-d) that, the multi-frequency method produces a phase error in pixel (180,42), while WFF-QG and WFRC-QG algorithm produce correct phase unwrapping results. For the details of the accuracy, we show the cross-section of $y=130$ and its enlarged view for phase unwrapping results of the three algorithms in Figs. 3.5(e) and Figs. 3.5(f). The red, black, blue and pink lines show the phase unwrapping results by the multi-frequency method, the WFF-QG algorithm, the WFRC-QG algorithm and the WFR-QG algorithm, respectively. Because the accuracy of the proposed two phase compensation methods are in the same order of magnitude, we only show the unwrapping result by the WFR algorithm with least squares fitting method for phase compensation. It is seen that by excluding the error pixels in multi-frequency method, the phase unwrapping results (in radians) are close to one another for the above four algorithms. However, a close view in Fig. 3.5(f) shows the improvement of accuracy for the WFRC algorithm over that of the WFR algorithm for an experimental fringe pattern.

3.6 Conclusion

In this Chapter, two methods to compensate the phase bias were proposed for the WFR algorithm. It was seen that the phase could be successfully compensated once the second order derivatives of the phase were estimated. To estimate the second order derivatives of the phase, the proposed first method depended on two frequency points and linear equations while the second used more than two frequency points and the least squares fitting on a convolution basis. Simulated as well as experimental fringe patterns were used to evaluate the accuracy of the proposed WFRC algorithm. It was shown that both the

proposed phase compensation methods effectively suppressed the bias produced by the WFR algorithm and improved the MSEs of the phase error by two orders of magnitude. Experiments also showed that for a quadratic phase or even cubic phase, the WFRC algorithm was better than the WFF algorithm for phase estimation. The WFRC algorithm enabled the WFRC-QG a more accurate PU algorithm than the WFR-QG algorithm. Furthermore, as an algorithm with high sensitivity to the edges, the WFRC-QG algorithm was shown more effective than the WFF-QG algorithm for some fringe patterns with long discontinuities. It should be noted that in [27] the WFF-QG algorithm was shown outperformed the WFR-QG algorithm for some complicated wrapped phase maps in MRI or InSAR by providing more reliable quality maps. Thus in different applications, the WFRC-QG and the WFF-QG algorithms can complement each other to obtain accurate phase unwrapping results.

Chapter 4 Statistical analysis for windowed Fourier ridges algorithm in fringe pattern analysis

4.1 Introduction

As discussed in Chapter 2, the MLE method for 2-D quadratic phase estimation requires a four dimensional search and is extremely time-consuming in implementation. By contrast, the WFRC algorithm reduces 4-D search to 2-D, which is more feasible for implementation. To study the accuracy of the WFRC algorithm for phase reconstruction, this Chapter focuses on the statistical analysis of the WFRC algorithm on local frequency and phase estimates when an EPF is disturbed by additive white Gaussian noise. The MSEs for local frequency and phase estimates by the WFRC algorithm will be derived and compared with the standard CRBs, respectively. Then the Monte Carlo simulations are used to verify the theoretical results. As reviewed in Chapter 2, there exist other similar algorithms for parameter estimation from an EPF such as the HAF [23, 33, 76] and the CPF algorithm [34, 69]. They will be briefly compared at the end of the Chapter. A paper based on this Chapter has been published in Applied Optics.

It is shown by both theoretical analysis and Monte Carlo simulations that, the WFRC algorithm is a suboptimal estimator because both MSEs of the local frequency and phase estimates are larger than their corresponding CRBs. Nevertheless, the phase estimate by the WFRC algorithm is very close to its CRB and is worse by at most 4.2 times when $c_2 \leq 0.02$ for a data length of 151. Moreover, Monte Carlo simulations show that the WFRC algorithm is slightly better than the WFF algorithm for phase estimate when the phase is quadratic. Here, we only address the analysis of the 1-D WFR algorithm, while its extension to 2-D version is straightforward. It is noted that only the linear equations method as phase compensation is adopted for statistical analysis in this chapter due to following two reasons, 1) the statistical results for the least squares fitting method depend on the size of input dataset for the least squares fitting method, which will make the statistical analysis even more complicated. 2) the MSEs introduced by the proposed two phase compensation methods are in the same order of magnitude, thus the linear equations method is enough for the statistical analysis of the WFRC algorithm.

4.2 1-D windowed Fourier ridges algorithm

For a 1-D EPF shown as follows,

$$f(x) = f_0(x) + n(x), \quad (4.1)$$

where $f_0(x)$ is an intrinsic signal and $n(x)$ is the noise. The intrinsic signal can be expressed as,

$$f_0(x) = b \exp[j\varphi(x)], \quad (4.2)$$

where $j = \sqrt{-1}$; b is the amplitude and assumed to be locally constant; $\varphi(x)$ is assumed to be the quadratic as,

$$\varphi(x) = c_0 + c_1x + 0.5c_2x^2, \quad (4.3)$$

with c_i as the i -th derivative of $\varphi(x)$ at $x=0$, I.E.,

$$c_i = \varphi^{(i)}(0), \quad i = 0, 1, 2. \quad (4.4)$$

The local frequency at x is defined as,

$$\omega(x) = \frac{d\varphi}{dx} = c_1 + c_2x. \quad (4.5)$$

For a noiseless EPF without the noise term,

$$f(x) = f_0(x). \quad (4.6)$$

The EPF represented by Eq. (4.6) undergoes a forward windowed Fourier transform as follows [41, 50],

$$Sf(u; \xi) = Sf_0(u; \xi) = \int_{-\infty}^{\infty} f(x)g(u-x)\exp(-j\xi x)dx, \quad (4.7)$$

where the window is selected as a Gaussian function,

$$g(x-u) = \frac{1}{(\pi\sigma^2)^{1/4}} \exp\left[-\frac{(x-u)^2}{2\sigma^2}\right], \quad (4.8)$$

with σ as its kernel size. Equation (4.7) indicates that harmonics with linear phase $\exp(j\xi x)$ are used to match an EPF with a quadratic phase $\exp[j\varphi(x)]$. Through some derivations [50], the following expressions can be obtained,

$$Sf(u; \xi) = A(u; \xi) \exp[j\Phi(u; \xi)], \quad (4.9)$$

where

$$A(u; \xi) = b \left(\frac{4\pi\sigma^2}{1 + \sigma^4 c_2^2} \right)^{1/4} \exp \left\{ -\frac{\sigma^2 [\xi - \omega(u)]^2}{2(1 + \sigma^4 c_2^2)} \right\}, \quad (4.10)$$

$$\Phi(u, \xi) = \varphi(u) - \xi u - \frac{\sigma^4 c_2 [\xi - \omega(u)]^2}{2(1 + \sigma^4 c_2^2)} + 0.5 \arctan(\sigma^2 c_2). \quad (4.11)$$

It is obvious that $|Sf(u; \xi)|$ achieve the maximum value when $\xi = \omega(u)$. Therefore, the following WFRC algorithm can be established,

$$\hat{\omega}(u) = \arg \max_{\xi} |Sf(u; \xi)| = \arg \max_{\xi} |Sf(u; \xi) \exp(j\xi u)|, \quad (4.12)$$

$$\hat{\varphi}(u) = \text{angle} \{ Sf[u; \hat{\omega}(u)] \exp[j\hat{\omega}(u)u] \} - 0.5 \arctan(\sigma^2 \hat{c}_2), \quad (4.13)$$

where $\hat{\omega}(u)$, $\hat{\varphi}(u)$ and $\hat{c}_2$ are estimates of $\omega(u)$, $\varphi(u)$ and c_2 , respectively. In Eqs. (4.12) and (4.13), $Sf(u; \xi) \exp(j\xi u)$ is used instead of $Sf(u; \xi)$, as the former is obtained directly in the implementation by convolution operations [41].

To reconstruct the phase in Eq. (4.13), a small bias $0.5 \arctan(\sigma^2 \hat{c}_2)$ needs to be estimated and deducted from $\text{angle} \{ Sf[u; \hat{\omega}(u)] \exp[j\hat{\omega}(u)u] \}$. It is seen that the key of the phase compensation is to estimate $\hat{c}_2$. According to Eq. (4.5), the local frequency is linear. If frequencies at two points u_1 and u_2 , I.E., $\hat{\omega}(u_1)$ and $\hat{\omega}(u_2)$, have been estimated, $\hat{c}_2$ can be further estimated using linear equations method as,

$$\hat{c}_2 = [\hat{\omega}(u_2) - \hat{\omega}(u_1)] / (u_2 - u_1), \quad (4.14)$$

For the intrinsic signal, since there is no noise, $\hat{c}_2 = c_2$, $\hat{\omega}(u) = \omega(u)$, and $\hat{\varphi}(u) = \varphi(u)$.

4.3 Estimation of local frequency and phase from a noisy signal

4.3.1 Local frequency estimation

In Eq. (4.1), when the EPF is perturbed by additive white Gaussian noise as,

$$n(x) = n_r(x) + jn_i(x), \quad (4.15)$$

where both $n_r(x)$ and $n_i(x)$ are assumed with a mean of zero and a variance of σ_n^2 . The windowed Fourier spectrum of $f(x)$ in Eq. (4.1) is

$$Sf(u; \xi) = Sf_0(u; \xi) + Sn(u; \xi), \quad (4.16)$$

where $Sf_0(u; \xi)$ is represented by Eq. (4.7), while

$$Sn(u; \xi) = \int_{-\infty}^{\infty} n(x)g(x-u)\exp(-j\xi x)dx, \quad (4.17)$$

When the WFR algorithm of Eqs. (4.12) and (4.13) is applied to the noisy signal $f(x)$, the estimated local frequency $\hat{\omega}(u)$ will produce an error $\delta\omega(u)$. Based on principle of the first order perturbation, the shift $\delta\omega(u)$ can be estimated according to Eq. (2.39). With $Sf_0(u; \xi)$ corresponding to $g_N(\Omega)$, $Sn(u; \xi)$ corresponding to $\delta g_N(\Omega)$, and $\hat{\omega}(u) = \omega(u) + \delta\omega(u)$ corresponding to $\Omega = \Omega_0 + \delta\Omega$, the shift $\delta\omega(u)$ can be derived from Eq. (2.39) as follows,

$$\delta\omega(u) \approx -\frac{B}{A} = -\frac{(1+\sigma^4 c_2^2)^{5/4}}{b\sigma^2 (4\pi\sigma^2)^{1/4}} \times \text{Im}\left(\exp\left\{j\left[c_0 - 0.5c_2 u^2 + 0.5\arctan(\sigma^2 c_2)\right]\right\} \int_{-\infty}^{\infty} n^*(x)g(x-u)(x-u)\exp(j\omega x)dx\right), \quad (4.18)$$

where $\text{Im}(\cdot)$ denotes the imagery part of the equations in the bracket. To obtain the above result, the following intermediate equations are needed,

$$Sf_0(u; \omega) = b \left(\frac{4\pi\sigma^2}{1+\sigma^4 c_2^2} \right)^{1/4} \exp\left\{j\left[c_0 - 0.5c_2 u^2 + 0.5\arctan(\sigma^2 c_2)\right]\right\}, \quad (4.19)$$

$$\frac{\partial Sf_0(u; \omega)}{\partial \xi} = -ju Sf_0(u; \omega), \quad (4.20)$$

$$\frac{\partial^2 Sf_0^*(u; \omega)}{\partial \xi^2} = -Sf_0^*(u; \omega) \left[u^2 + \frac{\sigma^2}{(1+j\sigma^2 c_2)} \right], \quad (4.21)$$

$$Sn^*(u; \omega) = \int_{-\infty}^{\infty} n^*(x)g(u-x)\exp(j\omega x)dx, \quad (4.22)$$

$$\frac{\partial Sn^*(u; \omega)}{\partial \xi} = j \int_{-\infty}^{\infty} n^*(x)g(u-x)x\exp(j\omega x)dx, \quad (4.23)$$

Because $n_r(x)$ and $n_i(x)$ are independent with zero-mean, it can be seen from Eq. (4.18)

that

$$E[\delta\omega(u)] = 0. \quad (4.24)$$

The covariance between the frequency shift at two points u_1 and u_2 can be derived from the formula,

$$E[\delta\omega(u_1)\delta\omega(u_2)] = \frac{1}{2} \left(E\left\{[\delta\omega(u_1) + \delta\omega(u_2)]^2\right\} - E\left\{[\delta\omega(u_1)]^2\right\} - E\left\{[\delta\omega(u_2)]^2\right\} \right). \quad (4.25)$$

Since

$$E[\delta\omega(u_1) + \delta\omega(u_2)] = 0. \quad (4.26)$$

We can derive the first term on the right side of the Eq. (4.25) that,

$$\begin{aligned} E\left\{\left[\delta\omega(u_1) + \delta\omega(u_2)\right]^2\right\} &= \text{Var}\left[\delta\omega(u_1) + \delta\omega(u_2)\right] = \frac{(1 + \sigma^4 c_2^2)^{5/2}}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \\ &\text{Var}\left\{\text{Re}\left\{j \int_{-\infty}^{\infty} n^*(x) g(x-u_1)(x-u_1) \exp[j\gamma(u_1, x)] dx + j \int_{-\infty}^{\infty} n^*(x) g(x-u_2)(x-u_2) \exp[j\gamma(u_2, x)] dx\right\}\right\}, \\ &= \frac{(1 + \sigma^4 c_2^2)^{5/2}}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \text{Var}\left\{\int_{-\infty}^{\infty} n_i(x) \left\{g(x-u_1)(x-u_1) \cos[\gamma(u_1, x)] + g(x-u_2)(x-u_2) \cos[\gamma(u_2, x)]\right\} dx \right. \\ &\quad \left. + \int_{-\infty}^{\infty} n_r(x) \left\{g(x-u_1)(x-u_1) \sin[\gamma(u_1, x)] + g(x-u_2)(x-u_2) \sin[\gamma(u_2, x)]\right\} dx\right\} \end{aligned} \quad (4.27)$$

where $n^*(x) = n_r(x) - jn_i(x)$ and $\gamma(u, x) = 0.5c_2u^2 - c_0 - 0.5\arctan(\sigma^2 c_2) - (c_2u + c_1)x$.

Since $n_r(x)$ and $n_i(x)$ are independent, according to the property of the variance that,

$$\text{Var}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \text{Var}(X_i). \quad (4.28)$$

Eq. (4.27) can be expended as,

$$\begin{aligned} E\left\{\left[\delta\omega(u_1) + \delta\omega(u_2)\right]^2\right\} &= \frac{(1 + \sigma^4 c_2^2)^{5/2}}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \\ &\left(\int_{-\infty}^{\infty} \left\{g(x-u_1)(x-u_1) \cos[\gamma(u_1, x)] + g(x-u_2)(x-u_2) \cos[\gamma(u_2, x)]\right\}^2 \text{Var}[n_i(x)] dx \right. \\ &\quad \left. + \int_{-\infty}^{\infty} \left\{g(x-u_1)(x-u_1) \sin[\gamma(u_1, x)] + g(x-u_2)(x-u_2) \sin[\gamma(u_2, x)]\right\}^2 \text{Var}[n_r(x)] dx\right), \\ &= \frac{(1 + \sigma^4 c_2^2)^{5/2}}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \frac{\sigma_n^2}{\sigma^2} \times \left\{\int_{-\infty}^{\infty} g^2(x-u_1)(x-u_1)^2 dx + \int_{-\infty}^{\infty} g^2(x-u_2)(x-u_2)^2 dx + \right. \\ &\quad \left.\int_{-\infty}^{\infty} 2g(x-u_2)(x-u_2)g(x-u_1)(x-u_1) \cos(0.5c_2u_1^2 - 0.5c_2u_2^2 + c_2u_2x - c_2u_1x) dx\right\} \end{aligned} \quad (4.29)$$

where $\text{Var}[n_r(x)] = \text{Var}[n_i(x)] = \sigma_n^2$.

Similarly, $E\left\{\left[\delta\omega(u_1)\right]^2\right\}$ and $E\left\{\left[\delta\omega(u_2)\right]^2\right\}$ can be derived as

$$E\left\{\left[\delta\omega(u_1)\right]^2\right\} = \frac{(1+\sigma^4 c_2^2)^{5/2} \sigma_n^2}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \int_{-\infty}^{\infty} g^2(x-u_1)(x-u_1)^2 dx, \quad (4.30)$$

$$E\left\{\left[\delta\omega(u_2)\right]^2\right\} = \frac{(1+\sigma^4 c_2^2)^{5/2} \sigma_n^2}{b^2 \sigma^4 (4\pi\sigma^2)^{1/2}} \times \int_{-\infty}^{\infty} g^2(x-u_2)(x-u_2)^2 dx. \quad (4.31)$$

Substitute Eqs. (4.29-4.31) and Eq. (4.8) into Eq. (4.25) and by simple integrations, we have

$$E\left[\delta\omega(u_1) \cdot \delta\omega(u_2)\right] \approx \frac{(1+\sigma^4 c_2^2)^{5/2}}{8\sqrt{\pi}\sigma^3 \text{SNR}} \left[1 - \frac{s^2(c_2^2\sigma^4 + 1)}{2\sigma^2}\right] \exp\left[-\frac{s^2(c_2^2\sigma^4 + 1)}{4\sigma^2}\right], \quad (4.32)$$

where $\cos(\varsigma) = \frac{1}{2}[\exp(j\varsigma) + \exp(-j\varsigma)]$ is used for derivation; $\text{SNR} = \frac{b^2}{2\sigma_n^2}$ and $s = u_2 - u_1$.

By setting $u_1 = u_2$, Eq. (4.32) becomes,

$$E\left\{\left[\delta\omega(u_1)\right]^2\right\} = E\left\{\left[\delta\omega(u_2)\right]^2\right\} = V_\omega \approx \frac{(1+\sigma^4 c_2^2)^{5/2}}{8\sqrt{\pi}\sigma^3 \text{SNR}}, \quad (4.33)$$

from which the MSE of the local frequency is seen independent of the location u . The MSE of $\hat{\omega}(u)$ increases with c_2 , which is intuitively understandable since the window Fourier basis with linear phase is used to match the signal with quadratic phase. It is also intuitive that the variance of $\hat{\omega}(u)$ decreases with SNR. From Eq. (4.33), the optimal value of σ that minimizes $E\left[(\delta\omega)^2\right]$ varies with c_2 and $\sigma=10$ is a good choice which is used throughout this Chapter.

4.3.2 Phase estimation

In this subsection, the MSE of the phase estimate is derived when the intrinsic signal is perturbed by additive white Gaussian noise. As seen from Section 4.2, the ridge locates at

$$\xi = \hat{\omega}(u) = c_1 + c_2 u + \delta\omega(u), \quad (4.34)$$

The windowed Fourier spectrum of Eq. (4.1) is

$$\begin{aligned} \hat{V} &= Sf[u; \hat{\omega}(u)] \exp[j\hat{\omega}(u)u] \\ &= Sf_0[u; \hat{\omega}(u)] \exp[j\hat{\omega}(u)u] + Sn[u; \hat{\omega}(u)] \exp[j\hat{\omega}(u)u]. \end{aligned} \quad (4.35)$$

Following the derivations in Refs [69, 76], the noises are assumed to be small for the first order perturbation analysis. Thus for the first term on the right hand of Eq. (4.35) regarding the intrinsic signal, the second order Taylor expansion of exponential function is used, while for the second term regarding noise which is small comparing with the intrinsic signal, the zero order Taylor expansion is adopted. These approximations lead Eq. (4.35) to the following equation,

$$\begin{aligned} \hat{V} &\approx Sf_0[u; \omega(u)] \exp[j\omega(u)u] + Sn[u; \omega(u)] \exp[j\omega(u)u] \\ &\quad - 0.5 \int_{-\infty}^{\infty} f_0(x) (\delta\omega)^2 (u-x)^2 \exp[j\omega(u-x)] g(u-x) dx, \end{aligned} \quad (4.36)$$

which can be further derived as

$$\hat{V} \approx V_0 (1 + \tau - \eta), \quad (4.37)$$

where

$$\begin{aligned} V_0 &= Sf_0[u; \omega(u)] \exp[j\omega(u)u] \\ &= b \left(\frac{4\pi\sigma^2}{1 + \sigma^4 c_2^2} \right)^{1/4} \exp \left\{ j \left[\varphi(u) + 0.5 \arctan(\sigma^2 c_2) \right] \right\}, \end{aligned} \quad (4.38)$$

$$\tau = Sn[u; \omega(u)] \exp[j\omega(u)u] / V_0, \quad (4.39)$$

$$\eta = \frac{\sigma^2 (1 + j\sigma^2 c_2)}{2(1 + \sigma^4 c_2^2)} [\delta\omega(u)]^2, \quad (4.40)$$

The following approximation can be used when D is small,

$$\log[C(1+D)] \approx \log C + D. \quad (4.41)$$

Taking $D = \tau - \eta$, we have

$$\log \hat{V} \approx \log \left[b \left(\frac{4\pi\sigma^2}{1+\sigma^4 c_2^2} \right)^{1/4} \right] + j \left[\varphi(u) + 0.5 \arctan(\sigma^2 c_2) \right] + \tau - \eta. \quad (4.42)$$

The phase of $\hat{V}$ is

$$\hat{\Phi} = \text{angle}(\hat{V}) = \text{Im}[\log(\hat{V})] \approx \varphi(u) + \text{Im}(\tau) - \text{Im}(\eta) + 0.5 \arctan(\sigma^2 c_2), \quad (4.43)$$

According to Eq. (4.13), $0.5 \arctan(\sigma^2 \hat{c}_2)$ is deducted from $\text{angle}(\hat{V})$ and thus,

$$\begin{aligned} \delta\varphi &\approx \text{Im}(\tau) - \text{Im}(\eta) + 0.5 \arctan(\sigma^2 c_2) - 0.5 \arctan(\sigma^2 \hat{c}_2) \\ &= \text{Im}(\tau) - \frac{\sigma^4 c_2}{2(1+\sigma^4 c_2^2)} [\delta\omega(u)]^2 - \frac{0.5\sigma^2 \delta c_2}{1+\sigma^4 c_2^2} \\ &= (\delta\varphi)_1 + (\delta\varphi)_2 + (\delta\varphi)_3 \end{aligned} \quad (4.44)$$

where $\arctan(a) - \arctan(b) = \arctan\left(\frac{a-b}{1+ab}\right)$, $\frac{\sigma^2 \delta c_2}{1+\sigma^4(c_2 \delta c_2 + c_2^2)} \ll 1$ and $\delta c_2 \ll c_2$ are used

for approximations. For simplicity, the three terms in the right hand of Eq. (4.44) are denoted as $(\delta\varphi)_1$, $(\delta\varphi)_2$, and $(\delta\varphi)_3$, respectively. In Eq. (4.44), the first term of phase error $(\delta\varphi)_1$ is caused by the additive noise, the second term $(\delta\varphi)_2$ is due to the local frequency error $\delta\omega(u)$, and $(\delta\varphi)_3$ is introduced by the phase compensation.

It is not difficult to see that $E[(\delta\varphi)_1] = E[(\delta\varphi)_3] = 0$. Thus according to Eq. (4.33),

$$E(\delta\varphi) \approx E[(\delta\varphi)_2] = E \left\{ -\frac{\sigma^4 c_2}{2(1+\sigma^4 c_2^2)} [\delta\omega(u)]^2 \right\} = -\frac{\sigma c_2 (1+\sigma^4 c_2^2)^{3/2}}{16\sqrt{\pi} \text{SNR}}. \quad (4.45)$$

The MSE of the phase consists of six terms is as follows,

$$E[(\delta\varphi)^2] = E[(\delta\varphi)_1^2] + E[(\delta\varphi)_2^2] + E[(\delta\varphi)_3^2] + 2E[(\delta\varphi)_1(\delta\varphi)_2] + 2E[(\delta\varphi)_2(\delta\varphi)_3] + 2E[(\delta\varphi)_1(\delta\varphi)_3] \quad (4.46)$$

Since $E[\text{Im}(\tau)] = 0$, the first term of Eq. (4.46) $E[(\delta\varphi)_1^2]$ can be derived using Eqs. (4.28)

and (4.8) as,

$$\begin{aligned} E[(\delta\varphi)_1^2] &= \text{Var}[\text{Im}(\tau)] = \left(\frac{1 + \sigma^4 c_2^2}{4\pi\sigma^2 b^4} \right)^{1/2} \text{Var} \left(\text{Im} \left[\int_{-\infty}^{\infty} S_n(x) g(u-x) \exp[jr(u,x)] dx \right] \right) \\ &= \left(\frac{1 + \sigma^4 c_2^2}{4\pi\sigma^2 b^4} \right)^{1/2} \text{Var} \left\{ \int_{-\infty}^{\infty} n_i(x) \cos[r(u,x)] g(u-x) dx + \int_{-\infty}^{\infty} n_r(x) \sin[r(u,x)] g(u-x) dx \right\} \\ &= \left(\frac{1 + \sigma^4 c_2^2}{4\pi\sigma^2 b^4} \right)^{1/2} \sigma_n^2 \int_{-\infty}^{\infty} g(u-x)^2 dx \\ &= \frac{(1 + \sigma^4 c_2^2)^{1/2}}{4\sqrt{\pi}\sigma\text{SNR}} \end{aligned} \quad (4.47)$$

By assuming the probability density function of $\delta\omega$ is Gaussian, the second term can be approximated as follows according to page 27 in Ref. [75],

$$E[(\delta\varphi)_2^2] = \frac{\sigma^8 c_2^2}{4(1 + \sigma^4 c_2^2)^2} E\{[\delta\omega(u)]^4\} \approx \frac{\sigma^8 c_2^2}{4(1 + \sigma^4 c_2^2)^2} 3 \left(E\{[\delta\omega(u)]^2\} \right)^2 = \frac{3\sigma^2 c_2^2 (1 + \sigma^4 c_2^2)^3}{256\pi(\text{SNR})^2} \quad (4.48)$$

When c_2 is estimated by Eq. (4.14), its error is,

$$\delta c_2 = [\delta\omega(u_2) - \delta\omega(u_1)] / (u_2 - u_1), \quad (4.49)$$

which has a mean of zero. By using Eqs. (4.32) and (4.49), the variance of δc_2 can be derived as,

$$E[(\delta c_2)^2] \approx \frac{2}{s^2} V_\omega \left\{ 1 - \left[1 - \frac{s^2}{2\sigma^2} (\sigma^4 c_2^2 + 1) \right] \exp \left[-\frac{s^2 (\sigma^4 c_2^2 + 1)}{4\sigma^2} \right] \right\}. \quad (4.50)$$

By using Eq. (4.50), the third term can be derived as,

$$E[(\delta\varphi)_3^2] = \frac{\sigma(1+\sigma^4 c_2^2)^{1/2}}{16\sqrt{\pi} s^2 \text{SNR}} \left\{ 1 - \left[1 - \frac{s^2}{2\sigma^2} (\sigma^4 c_2^2 + 1) \right] \exp \left[-\frac{s^2 (\sigma^4 c_2^2 + 1)}{4\sigma^2} \right] \right\}. \quad (4.51)$$

The term $E[(\delta\varphi)_1(\delta\varphi)_2]$ can be expressed as the following formula that

$$E[(\delta\varphi)_1(\delta\varphi)_2] = E \left[\left(\sum_x m_1(x) n_r(x) + m_2(x) n_i(x) \right) \left(\sum_x m_3(x) n_r(x) + m_4(x) n_i(x) \right)^2 \right], \quad (4.52)$$

where $\sum$ denotes the integral, $n_r(x)$ and $n_i(x)$ are real and imagery part of the independent Gaussian random noise. $m_1(x) \sim m_4(x)$ are their constant coefficients.

Eq.(4.52) can be expended as,

$$\begin{aligned} E[(\delta\varphi)_1(\delta\varphi)_2] &= E \left[\left(\sum_x m_1(x) n_r(x) \right) \left(\sum_x m_3(x) n_r(x) \right)^2 \right] + \\ &\quad E \left[\left(\sum_x m_2(x) n_i(x) \right) \left(\sum_x m_4(x) n_i(x) \right)^2 \right] + \\ &\quad 2E \left[\left(\sum_x m_2(x) n_i(x) \right) \left(\sum_x m_3(x) n_r(x) \right) \left(\sum_x m_4(x) n_i(x) \right) \right] + \\ &\quad 2E \left[\left(\sum_x m_1(x) n_r(x) \right) \left(\sum_x m_3(x) n_r(x) \right) \left(\sum_x m_4(x) n_i(x) \right) \right] \end{aligned} \quad (4.53)$$

Since $n_r(x)$ and $n_i(x)$ are independent,

$$E[(\delta\varphi)_1(\delta\varphi)_2] = M_1 E \left\{ [n_r(x)]^3 \right\} + M_2 E \left\{ [n_i(x)]^3 \right\}, \quad (4.54)$$

where M_1 and M_2 are sum of the constant coefficients. Furthermore, because $n_r(x)$ and

$n_i(x)$ follow the normal distribution, $E \left\{ [n_r(x)]^3 \right\} = E \left\{ [n_i(x)]^3 \right\} = 0$. Thus,

$$E[(\delta\varphi)_1(\delta\varphi)_2] = 0. \quad (4.55)$$

Because of the symmetric distribution of $\delta\omega$, $E[(\delta\varphi)_2(\delta\varphi)_3]$ can be evaluated to be zero (Ref. [75], page 27).

For the last term, the covariance between $(\delta\varphi)_1$ and $(\delta\varphi)_3$ can be derived using Eq. (4.25).

Since $E[(\delta\varphi)_1 + (\delta\varphi)_3] = 0$, we have

$$E\left\{[(\delta\varphi)_1 + (\delta\varphi)_3]^2\right\} = \text{Var}[(\delta\varphi)_1 + (\delta\varphi)_3] = \frac{(1 + \sigma^4 c_2^2)^{1/2}}{4s^2 b^2 (4\pi\sigma^2)^{1/2}} \times \left[\text{Var}\left(\int_{-\infty}^{\infty} n_i(x) \{2sg(u-x)\cos[\gamma(u,x)] + g(u_1-x)(x-u_1)\cos[\gamma(u_1,x)] - g(u_2-x)(x-u_2)\cos[\gamma(u_2,x)]\}dx\right) + \right. \\ \left. \text{Var}\left(\int_{-\infty}^{\infty} n_r(x) \{2sg(u-x)\sin[\gamma(u,x)] + g(u_1-x)(x-u_1)\sin[\gamma(u_1,x)] - g(u_2-x)(x-u_2)\sin[\gamma(u_2,x)]\}dx\right) \right] \quad (4.56)$$

According to Eqs. (4.25) and (4.28), Eq. (4.56) can be derived as,

$$E[(\delta\varphi)_1(\delta\varphi)_3] = \frac{\sqrt{1 + \sigma^4 c_2^2}}{4s^2 4\sqrt{\pi}\text{SNR}\sigma} \times \left\{ 4s \int_{-\infty}^{\infty} g(u-x)g(u_1-x)(x-u_1)\cos[\gamma(u,x) - \gamma(u_1,x)]dx - \right. \\ \left. 4s \int_{-\infty}^{\infty} g(u-x)g(u_2-x)(x-u_2)\cos[\gamma(u,x) - \gamma(u_2,x)]dx \right\} \quad (4.57)$$

Also using $\cos(\varsigma) = \frac{1}{2}[\exp(j\varsigma) + \exp(-j\varsigma)]$, Eq. (4.57) can be derived as

$$E[(\delta\varphi)_1(\delta\varphi)_3] = \frac{(1 + \sigma^4 c_2^2)^{1/2}}{16\sqrt{\pi}\sigma s \text{SNR}} \left\{ (u-u_1)\exp\left[-\frac{(u-u_1)^2(1 + \sigma^4 c_2^2)}{4\sigma^2}\right] - (u-u_2)\exp\left[-\frac{(u-u_2)^2(1 + \sigma^4 c_2^2)}{4\sigma^2}\right] \right\}. \quad (4.58)$$

Consequently, the MSE of phase estimate can be summarized as,

$$E[(\delta\varphi)^2] \approx \frac{(1 + \sigma^4 c_2^2)^{1/2}}{4\sqrt{\pi}\sigma \text{SNR}} [1 + k_1(c_2, \text{SNR}, \sigma) + k_2(c_2, \sigma, s) + k_3(c_2, \sigma, s, l)], \quad (4.59)$$

with

$$k_1(c_2, \text{SNR}, \sigma) = \frac{3\sigma^3 c_2^2 (1 + \sigma^4 c_2^2)^{5/2}}{64\sqrt{\pi} \text{SNR}}, \quad (4.60)$$

$$k_2(c_2, \sigma, s) = \frac{(1 + \sigma^4 c_2^2)^2}{s^2 \sigma^2} \left\{ 1 - \left[1 - \frac{s^2}{2\sigma^2} (1 + \sigma^4 c_2^2) \right] \exp \left[-\frac{s^2 (1 + \sigma^4 c_2^2)}{4\sigma^2} \right] \right\}, \quad (4.61)$$

$$k_3(c_2, \sigma, s, l) = \frac{1}{4s} \left\{ l \exp \left[-\frac{l^2 (1 + \sigma^4 c_2^2)}{4\sigma^2} \right] - (l - s) \exp \left[-\frac{(l - s)^2 (1 + \sigma^4 c_2^2)}{4\sigma^2} \right] \right\}. \quad (4.62)$$

where $l = u - u_1$ and $s = u_2 - u_1$.

Given an EPF with a data length of N , $c_2 \leq \pi/N$ is required so that the signal is not aliased. For example, $c_2 \leq 0.02$ for $N = 151$. With this data length, simulation shows that both $k_1(c_2, \text{SNR}, \sigma)$ and $k_2(c_2, \sigma, s)$ are ignorable when $c_2 \leq 0.02$ and $\sigma \leq 10$. As for $k_3(c_2, \sigma, s, l)$, it contributes up to about 0.1 comparing with 1 in Eq. (4.59) when $s \geq 50$, thus it can also be ignorable. Overall, Eq. (4.59) is approximated as,

$$E[(\delta\varphi)^2] \approx \frac{(1 + \sigma^4 c_2^2)^{1/2}}{4\sqrt{\pi} \sigma \text{SNR}}, \quad (4.63)$$

which depends on parameters of c_2 , σ and SNR. Similarly to $E[(\delta\omega)^2]$, $E[(\delta\varphi)^2]$ increases with c_2 and decreases with SNR. The bias expressed by Eq. (4.45) is small comparing with the MSE represented by Eq. (4.63). Thus the MSE and variance of $\delta\varphi$ are approximately the same.

4.3.3 Comparison with the Cramér-Rao Bounds

The previous analysis focuses on the continuous signal but can be seen as an approximation also for discrete data. To emphasize the units of various parameters: rad stands for $\varphi(x)$, rad/pixel stands for $\omega(x)$, $\text{rad}/(\text{pixel})^2$ stands for c_2 , and pixel stands for σ . They are omitted since no confusion will be raised.

For a window size σ , the Gaussian window should be theoretically infinitely long. It is truncated to $6\sigma+1$ in Ref. [41]. In this Chapter, it is truncated to $10\sigma+1$ for better verification between theoretical analysis and Monte Carlo simulations. The truncation affect between two different truncations will be compared in Section 4.4. In the WFR algorithm, a sliding window approach is adopted. The data within $[u-5\sigma, u+5\sigma]$ are used to estimate the local frequency at u by a WFR. The data length is thus $10\sigma+1$. In order to compensate the phase error, the data within $[u, u+10\sigma]$ is used for another WFR to estimate the frequency at $u+5\sigma$. By using these two frequency estimates, $\hat{c}_2$ can be computed by Eq. (4.14) and the phase can be further compensated by Eq. (4.13). Thus the data involved in phase estimation will be with the length of $15\sigma+1$.

The CRBs for local frequency and phase estimation have been derived in Ref. [194] and summarized in Table 4.1 given the above data length. The ratios of the MSEs by the WFR algorithm over the CRBs are also shown in Table 4.1. According to the theoretical results shown in Table 4.1, the WFR results are larger than the CRBs and thus they are suboptimal. The ratio MSE/CRB of both local frequency and phase estimates increase with c_2 . Two examples are given in Table 4.1 with $(c_2 = 0.001, \sigma = 10)$ and $(c_2 = 0.02, \sigma = 10)$, respectively. Although the MSE of the local frequency estimate is not close to its

CRB, the MSE of the phase estimate is, which are shown in Table 4.1. This is because the phase error caused by the local frequency estimate error $(\delta\varphi)_2$, as given in Eq. (4.44), is not significant comparing with the error directly caused by the noise.

Table 4.1 Theoretical MSEs of local frequency and phase estimates by the WFR and their CRBs.

Reprint from Ref. [195] with permission from Optical Society of America.

	CRB	MSE/CRB	MSE/CRB ($c_2 = 0.001, \sigma = 10$)	MSE/CRB ($c_2 = 0.02, \sigma = 10$)
Local frequency estimate	$\frac{6}{(10\sigma + 1)^3 \text{SNR}}$	$11.75(1 + \sigma^4 c_2^2)^{5/2}$	12	656.85
Phase estimate	$\frac{1.125}{(15\sigma + 1)\text{SNR}}$	$1.88(1 + \sigma^4 c_2^2)^{1/2}$	1.89	4.20

4.4 Simulations

Monte Carlo simulations will be carried out in this section to (i) verify the above theoretical analysis, (ii) to compare local frequency and phase estimates with corresponding CRBs numerically, (iii) to give an intuitive illustration about the effectiveness of the WFRC algorithm, and (iv) to compare the WFRC algorithm with its sister algorithm WFF and other model based phase estimation algorithms.

4.4.1 Monte Carlo simulations and comparison with CRBs

Figure 4.1 Simulated wrapped phase maps (a) $c_2 = 0.001$ (b) $c_2 = 0.02$. Reprint from Ref. [195] with permission from Optical Society of America.

Figure 4.2 Theoretical and experimental MSEs by the WFRC algorithm

(a) Local frequency estimate (b) Phase estimate . Reprint from Ref. [195] with permission from Optical Society of America.

The 1-D WFRC algorithm is applied to a quadratic phase field contaminated by additive white Gaussian noise with the parameters $b=1$, $c_0=1$, $c_1=4\times 10^{-3}$, $c_2=1\times 10^{-3}$ and $N=151$. Such a simulated phase, without noise, is shown in Fig. 4.1(a). The SNRs of the signal varies from -10dB to 20dB with a 1dB interval. For each SNR value, 500 Monte Carlo simulations are performed. The parameters of the WFRC algorithm are set as follows: the frequency is searched from -1 to 1 with a step of 0.00001 , and the size of Gaussian window is $\sigma=10$.

As shown in Fig. 4.2, the achievable MSEs of the WFRC algorithm is compared with the corresponding CRBs. The solid lines depict the CRBs; the dash-dot lines indicate the theoretically predicted MSEs; and the plus sign markers denote the measured MSEs. The measured and theoretical MSEs agree very well for both local frequency and phase estimates. Furthermore, they are all suboptimal compared with their CRBs, which has been revealed by previous theoretical analysis. It is seen from the Fig. 4.2 that for the local frequency estimate, the MSE is higher than its CRB by less than 12 times (about 10dB), while for the phase estimate, the MSE is higher than its CRB by only less than 2 times (about 3dB), which show that the measured MSEs are consistent with Table 4.1. For instance, the efficiency of the phase estimate, I.E., CRB/MSE , is about 50% , which is not high. However, it can be seen from Fig. 4.2(b) that the MSE at 5dB is less than 0.01 rad^2 , which is satisfactory in many applications. Since the threshold SNR below which the MSE dramatically increases is often used to characterize an algorithm, we analysis the threshold SNR for the experiments. It can be observed from Figs. 4.2(a) and 4.2(b) that

for the data with $N=151$, the threshold SNR is about -5dB for frequency estimate and about -3dB for phase estimate, respectively.

Figure 4.3 The influence of c_2 in the WFRF algorithm (a) Local frequency estimate (b) Phase estimate. Reprint from Ref. [195] with permission from Optical Society of America.

4.4.2 Influences of c_2 and window truncation

In section 4.3 we showed that the MSEs of the frequency and phase estimates by the WFRF algorithm are affected by c_2 , which is also verified by the Monte Carlo simulations. In Figs. 4.3(a) and 4.3(b), the solid lines are CRBs; the plus sign, asterisk and diamond markers show the measured MSEs when c_2 are 0.001, 0.01 and 0.02, respectively. It can be seen in Fig. 4.1(b) that the phase changes very fast when $c_2 = 0.02$. Theoretical MSEs are not shown in Fig. 4.3 because they agree well with the measured ones. Figs. 4.3(a) and (b) show that when c_2 increases from 0.001 to 0.02, the MSE of the

local frequency estimate increases by 54 times (about 17dB), but the MSE of the phase estimate increases by only about 2 times (about 3dB). Thus the phase estimate is seen to be more robust with the increment of c_2 . Furthermore, the threshold SNR is increased to 0dB for the local frequency estimate and 3dB for the phase estimate when the given the data length 151. It is worth mentioning that with the development of the detectors, data can be acquired with higher resolution and thus lower c_2 value.

Figure 4.4 The influence of window truncation, $6\sigma+1=61$ vs. $10\sigma+1=101$. Reprint from Ref. [195] with permission from Optical Society of America.

Since the size of the Gaussian window is truncated to $6\sigma+1$ instead of $10\sigma+1$ in real practices, it is of interest to see the impact by the truncated window size for the WFR algorithm. The SNR of the signal varies from -10dB to 100dB with a 5dB interval. It is shown in Fig. 4.4 that the Monte Carlo simulations of the WFR algorithm with window sizes of $6\sigma+1=61$ and $10\sigma+1=101$ for the WFR are indicated as plus signs markers and diamond markers, respectively. Comparing with the truncated size $10\sigma+1=101$, the

truncated size $6\sigma + 1 = 61$ contributes an MSE of 10^{-6} rad^2 (-60dB), which is considered not significant in practice.

Figure 4.5 Phase estimate by the WFR and WFF algorithms. Reprint from Ref. [195] with permission from Optical Society of America.

4.4.3 WFR vs. WFF

The WFR algorithm is also compared with its sister algorithm WFF by Monte Carlo simulations. The description of the WFF has been documented in detail in [50] and will be introduced in later Chapter, thus it is omitted to make the Chapter concise. The signal simulated in Section 4.4.1 is used again in the experiments. The input parameters of the WFF algorithm are as follows, the frequency varies from -1 to 1 with a step of 0.00001 ; the size of Gaussian window is $\sigma=10$; and the threshold is set as 3 . The truncated Gaussian window size for the WFF algorithm is selected as $4\sigma+1$ since the larger window size does not improve the result significantly [50]. Figure 4.5 shows the Monte

Carlo simulations of the phase estimate by the WFRC and WFF algorithms, where solid lines are CRBs, plus signs markers indicate the measured MSEs by the WFRC algorithm; and diamond markers show the measured MSEs by the WFF algorithm. It can be seen that the WFF algorithm for phase estimate is also close to the CRB and slightly worse than the WFRC algorithm. This is because the WFRC retrieves parameter from the highest spectrum coefficient, I.E., the ridge, while the WFF utilizes the coefficients higher than a threshold but could be lower than the ridge.

4.4.4 Comparing with other model based phase estimation methods

As briefly introduced in Chapter 2, the HAF and the CPF are two typical algorithms for parameter estimation from an EPF. Both of them need a preprocessing procedure to decouple the parameters to simplify the optimization problem [69, 76]. The preprocessing procedure reduces the robustness of these two algorithms and makes the estimation more difficult especially when the input signal is very noisy. By comparison, the WFRC algorithm processes the original signal directly and is thus simpler. Comparing with the HAF and CPF, the WFRC algorithm produces larger MSEs, but it can process more noisy signals successfully. For instance, to process the same signal as in Section 4.4.1, the threshold SNR for both HAF and CPF algorithm is 0dB while the threshold SNR for the WFRC algorithm is -3dB, which is shown in Fig. 4.6. Although the threshold SNR for the WFRC algorithm is also depend on c_2 , it can be seen from the experiments that when

$c_2 < 0.01$, the threshold SNR for the WFRC algorithm is less than both of the HAF and the CPF algorithm. For large c_2 (I.E. $c_2 \geq 0.01$), congruence operation can be adopted to make the WFRC algorithm optimal [35].

Figure 4.6 Comparison of phase estimate by the WFRC algorithm, HAF and CPF algorithms.

Reprint from Ref. [195] with permission from Optical Society of America.

4.5 Conclusion & Discussions

This Chapter focused on the achievable accuracy of the WFRC algorithm for the local frequency and phase estimates. Statistical analysis was given for the 1-D WFRC algorithm when the quadratic exponential phase field was contaminated by additive white Gaussian noise, while the Monte Carlo simulations were used to verify the theoretical analysis. The MSEs for local frequency and phase estimates depended on the second order derivative of the phase, the Gaussian window size, and the SNR.

The theoretical analysis and Monte Carlo simulations showed that the MSEs for the local frequency and phase estimates were suboptimal comparing with the corresponding CRBs. When c_2 increased, the MSEs and threshold SNRs for both local frequency and phase estimates increased accordingly. However, the MSE for the phase estimates was close to the CRB. Both the comparison with the WFF algorithm and other model based phase estimate algorithms showed that the WFRC was a robust algorithm with comparable high accuracy and lower threshold SNR.

The statistical analysis for the 2-D WFRC algorithm was not given to make the chapter concise. However, its theoretical MSEs for local frequency and phase estimates were derived and verified by Monte Carlo simulations after submitting the thesis, which have been published in a conference paper [196]. Similar results can be observed for the 2-D WFRC algorithm as the 1-D WFRC algorithm.

Chapter 5 Accelerating the WFT-based algorithms using multicore CPU in MATLAB[®] environment

5.1 Introduction

It was shown by the previous Chapters that the WFT-based algorithms could achieve very high accuracy for phase retrieval. However, they are computationally expensive and cost 45s to 3 min by Pentium-IV 3.2GHz desktop [41]. Since the WFT-based algorithms are highly parallelizable, they can be accelerated by parallel hardware very efficiently. Based on the principle of parallel computing, multicore has been the mainstream computing model in the computer market nowadays. Recently, Intel even released 80 cores processor called Teraflops Research Chip [197], which can overwhelm a supercomputer in the aspect of floating point operations per second. However, as we mentioned in Chapter 2, the former sequential programs or algorithms are hard to be automatically adapted into the multicore CPU. Thus additional parallel coding technique should be employed to parallelize them. MATLAB[®] [198] is among the first batch of software that support programming multicore CPU. As a high level programming language, the code of MATLAB[®] is concise, portable and easy to maintain. Using parallel computing toolbox

[59] of MATLAB[®] based on multicore CPU is an intuitive and simple way to improve the speed performance of the WFT-based algorithms. Furthermore, the tasks of this Chapter include:

- 1) To pave the way for the later heterogeneous system by understanding multicore CPU in the acceleration of the WFT based algorithms.
- 2) To use multicore CPU and MATLAB[®] as a prototyping model for the acceleration of the WFT-based algorithms in other parallel hardware such as GPU.
- 3) Since the multicore CPU is widely used in commodity PC, when other parallel hardware are not available, the proposed work can actually be used as a stand-alone acceleration component for fringe pattern analysis.

Based on the above reasons, this Chapter explores using multicore CPU and MATLAB[®] parallel computing toolbox to accelerate the WFT-based algorithms. For the WFF algorithm, the spectrums for different frequencies are thresholded and accumulated to form a final result, in which the computation of different spectrums have no communication except for a simple addition. This simple addition satisfies the rule of MATLAB[®] parallel computing toolbox and belongs to reductions [199] in which a value depends on all the iterations together, but independent of the iteration order. Thus in MATLAB[®] parallel computing toolbox, “parfor” command can be simply adopted instead of “for” to accelerate the WFF algorithm by dispatching computation loops for different frequencies into different cores of CPU, which can be denoted as task-parallel method. While for the WFR algorithm, the spectrums for different frequencies are compared to obtain a optimum result. The comparison among different spectrums involves complicated communications, which may cause heavy overhead thus not be suitable for

being accelerated by the task-parallel method. Thus we use data-parallel method to accelerate the WFR algorithm by dividing the input fringe pattern into several stripes which can be distributed into different cores of CPU for simultaneous processing. The rest of the Chapter is organized as follows. MATLAB[®] parallel computing toolbox are briefly introduced in Section 5.2; Using MATLAB[®] parallel computing toolbox and multicore CPU technique to accelerate the WFT-based algorithms is presented in Section 5.3; The results show that the parallel versions of WFT-based algorithms with simple modifications achieve the speedup up to 6.6 times, which is given in Section 5.4. The Chapter is concluded in Section 5.5. This Chapter is partially grounded on our paper “Parallel computing for fringe pattern processing: A multicore CPU approach in MATLAB[®] environment” which has been published in the journal of “Optics & Lasers in Engineering” [200].

5.2 MATLAB[®] parallel computing toolbox

Parallel computing toolbox provides a high-level structure for solving computationally expensive problems making use of multicore or bus connected computers to share the computation load by parallel processing. Parallel computing toolbox was first released in September 2006 as a part of MATLAB[®] 2006b and can support many workers or worker sessions [201] locally on a multicore desktop. Collaborating with the MATLAB[®] distributed computing server, parallel computing toolbox also supports cluster-based applications that use any scheduler or any number of workers on a computer cluster. Furthermore, converting former sequential MATLAB[®] code to parallel applications only

needs a few modifications of code and no programming in a low-level communication driver. These features of parallel computing toolbox enable scientists and engineers focusing on their algorithm design while enjoy the benefits of speedup using multicore CPU or a cluster of workstations.

Figure 5.1 Architecture using parallel computing toolbox and multicore CPU. Reprint from Ref. [200] with permission from Elsevier.

Parallel computing toolbox usually supports accelerating task-parallel and data-parallel problems. Moreover, the task parallel in parallel computing toolbox is restricted to embarrassing problems in which “iterations” of a parallel loop have little communication among them. By simply replacing “for” in former MATLAB[®] code by “parfor”, parallel computing toolbox automatically manages data and code transfer between the MATLAB[®] client session and the worker sessions. If parallel computing toolbox fails to detect the

presence of the worker sessions, it automatically reverts to sequential execution. Data-parallel problems in MATLAB[®] parallel computing toolbox mainly refer to the algorithms that require large amount of dataset to be processed. Parallel computing toolbox provides distributed arrays, parallel functions and uses “spmd” keyword for parallel execution in several worker sessions. When the dataset is divided into smaller pieces and sent to different cores, some non-edge pixels in the original dataset become edges pixels of these smaller datasets. Consequently, edge effects will be introduced. Because the data-parallel commands in MATLAB[®] parallel computing toolbox do not provide edge consistency function, the data-parallel commands will not be used in this Chapter. Thus we propose an alternative method to solve this problem, which will be introduced in Section 5.3.3.

In Fig. 5.1, we show the structure of parallel computing toolbox as a superstructure over other toolboxes and MATLAB[®] environment. For example, if 4 worker sessions have been opened, the parallel computing toolbox will automatically reserve the resource and underlying hardware cores of CPU. In windows task manager, five MATLAB[®] processes will show including one client session and 4 worker sessions. Worker sessions will be in idle status unless parallel instructions are called.

5.3 Using parallel computing toolbox and multicore CPU to accelerate the WFT-based algorithms

5.3.1 Windowed Fourier transform based algorithms

In this subsection, we discuss the implementation issue of the WFF and WFR algorithms as well as their parallelizability. Given the fact that the energy of noise is evenly distributed in the whole spectrum domain while the signal energy is only located in a specific spectrum, the windowed Fourier spectrums can be thresholded by setting small amplitude to zero to eliminate the noise. Then the filtered signal can be recovered by sum of the inverse WFT of thresholded spectrums. Based on Eqs. (2.22) and (2.23), the 2-D WFF algorithm can be implemented by convolution operations as,

$$\bar{f}(x, y) = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \left[\overline{f(u, v) \otimes h_{\xi, \eta}(u, v)} \right] \otimes h_{\xi, \eta}(x, y) d\xi d\eta. \quad (5.1)$$

where $\bar{f}(x, y)$ is the filtered result; $h_{\xi, \eta}(x, y) = g(x, y)e^{j\xi x + j\eta y}$; $g(x, y)$ is a symmetric window function; and $\otimes$ denotes the convolution operator. It is shown that in Eq. (5.1) the signal is convoluted with $h_{\xi, \eta}(x, y)$ for different frequencies ξ and η . Convolutions with different frequencies are independent of one another. This feature reveals the possibility to parallelize the algorithm by executing different convolutions in different

cores. Parallelizing the WFF algorithm using multicore CPU and parallel computing toolbox is addressed in Section 5.3.2.

For the 2-D WFR algorithm, fringe pattern is transformed to the time-frequency spectrum by a forward WFT according to Eq. (2.22). Then the spectrum amplitudes for different frequencies are compared with one another to get the largest value. The 2-D WFR algorithm can be implemented by the following equations,

$$[\omega_x(u, v), \omega_y(u, v)] = \arg \max_{\xi, \eta} |f(u, v) \otimes h_{\xi, \eta}(u, v)|, \quad (5.2)$$

$$r(u, v) = |f(u, v) \otimes h_{\omega_x(u, v), \omega_y(u, v)}(u, v)|, \quad (5.3)$$

$$\varphi(u, v) = \text{angle} \{ f(u, v) \otimes h_{\omega_x(u, v), \omega_y(u, v)}(u, v) \}, \quad (5.4)$$

where $r(u, v)$ is the retrieved windowed Fourier ridges map; $[\omega_x(u, v), \omega_y(u, v)]$ are the frequency locations which the windowed Fourier ridges appear; and the $\varphi(u, v)$ is the retrieved phase map. It is seen in Eq. (5.2) that a maximum ridge value should be searched from the frequency domain. This involves many communications among the cores of a CPU, which cannot be accelerated by simple task-parallel method in MATLAB[®] with the “parfor” operation. However, like the WFF algorithm, the WFR algorithm is a local processing algorithm, in which the processing of one pixel is independent of the processing of pixels outside a window. Thus the data-parallel method can be used based on the MATLAB[®] parallel computing toolbox and multicore CPU to accelerate the WFR algorithm, which are discussed in detail in Section 5.3.3.

5.3.2 Accelerating the WFF algorithm

```
function g=wff(f,sigma,wxl,wxi,wxh,wyl,wyi,wyh,thr)

s=round(2*sigma);
[y x]=meshgrid(-s:s,-s:s);
w=exp(-(x.*x+y.*y)/2/sigma/sigma);
w=w/sqrt(sum(sum(w.*w)));

g=f*0;
for wyt=wyl:wyi:wyh
    for wxt=wxl:wxi:wxh
        wave=w.*exp(j*wxt*x+j*wyt*y);
        sf=conv2(f,wave,'same');
        sf=sf.*(abs(sf)>=thr);
        g=g+conv2(sf,wave,'same');
    end
end
g=g/4/pi/pi*wxi*wyi;
```

(a)

```
function g=wff(f,sigma,wxl,wxi,wxh,wyl,wyi,wyh,thr)

s=round(2*sigma);
[y x]=meshgrid(-s:s,-s:s);
w=exp(-(x.*x+y.*y)/2/sigma/sigma);
w=w/sqrt(sum(sum(w.*w)));

g=f*0;
for wyt=wyl:wyi:wyh
    parfor wxT=0:(floor((wxh-wxl)/wxi))%changed line 1
        wxt=wxT*wxi+wxl;% changed line 2
        wave=w.*exp(j*wxt*x+j*wyt*y);
        sf=conv2(f,wave,'same');
        sf=sf.*(abs(sf)>=thr);
        g=g+conv2(sf,wave,'same');
    end
end
g=g/4/pi/pi*wxi*wyi;
```

(b)

Figure 5.2 MATLAB® code for sequential and task-parallel approach on WFF algorithm

(a) Codes for sequential WFF algorithm (b) Codes for task-parallel WFF algorithm. Reprint from Ref.

[200] with permission from Elsevier.

As mentioned in Section 5.2, parallel computing toolbox provides a method for task parallelism of embarrassing applications. By using “parfor” instead of “for”, we take the advantage of inherent schedule and memory management mechanism provided by parallel computing toolbox to accelerate the WFF algorithm. MATLAB[®] codes for both sequential and parallel version of the WFF algorithm are shown in Fig. 5.2. In practice, we use fast Fourier transform (FFT) to implement “conv2” in both sequential and parallel WFF algorithm. Among input parameters of the WFF algorithm, “f” is the input fringe pattern to be filtered; “sigma” indicates Gaussian window size; “wxl” (resp. “wyl”) is lower frequency bound along x (resp. y); “wxi” (resp. “wyi”) is an incremental step of the frequency along x (resp. y); “wxh” (resp. “wyh”) is upper frequency bound along x (resp. y); “thr” is a threshold; “g” is the filtered fringe pattern [41]. As highlighted in Fig. 5.2(b), only two lines are modified between the sequential and parallel version of the MATLAB[®] code: 1) “for” in sequential WFF code is replaced by “parfor” in parallel WFF code. 2) According to instructions of parallel computing toolbox [59], incremental step of “parfor” must be an integer. Hence, sequential code should be modified correspondingly for “parfor” incremental step, which is shown in Fig. 5.2(b). Former sequential “for loop” is automatically divided into N (from 2 to 8) parallel “for loop”, each of which will be executed in a different MATLAB[®] worker session.

To enable parallel computing toolbox, the MATLAB[®] command “matlabpool” should be invoked in MATLAB[®] command window. One can also specify the number of cores N using “matlabpool open N ”. Then resources such as CPU cores and memories are reserved automatically for different workers called MATLAB[®] labs. Correspondingly, “matlabpool close” command is used to release the reserved resources and disable parallel

computing toolbox. Note that “parfor” in MATLAB[®] cannot be nested in the same function [201]. In our parallel WFF algorithm, up to 8 different frequencies are processed by available cores simultaneously. Parallel tasks are automatically distributed to different cores of the CPU. One can benefit from the speedup provided by the parallel computing toolbox and multicore CPU while need not bother to manage the memory and schedule issues among CPU cores.

5.3.3 Accelerating the WFR algorithm

In data parallelism, if data elements have no dependencies among them in the process, then the same process can be applied to different data elements at the same time. This is also applicable for larger data segments without dependencies among them. Since the WFR is a local processing algorithm, in which processing local data is independent of that of others, the EPF can be segmented into several stripes and processed by different cores of CPU simultaneously.

A general parallel wrapping function is designed for local processing algorithms such as WFR. The function is written in MATLAB[®] “m file”, which can call any local image processing algorithm written as MATLAB[®] functions. The input EPF is divided into image strips along columns or rows, and then fed into the CPU cores. In each MATLAB[®] worker session, a same copy of the algorithm processes a different image strip. Then, results from different cores are transmitted back to the client session and combined to form the final result, which is the same as the results obtained by sequential computing. It

is noted that the transmission of image strips between client session and worker sessions is handled by parallel computing toolbox automatically.

```
'N' is the number of cores;  
'f' is the input fringe pattern;  
'seqFun' is the sequential function to be parallelized;  
it has the form of output=seqFun(f,paramters);  
'parameters' is a parameter array for seqFun;  
'overlapsize' is overlapping size for image consistency  
  
parfun(N,seqfun,f,overlapsize,parameters)  
  
if N==1  
    processedimage = seqfun(f,parameters)  
else  
    cut image into N strips along the row  
    take into consideration of overlapping size  
    parfor i = 1:N  
        stripresult(i) = seqFun(strip(i),parameters)  
    end  
    processedImage = combination of stripresults  
end
```

Figure 5.3 Pseudo code for data-parallel wrapping function.

Reprint from Ref. [200] with permission from Elsevier.

Figure. 5.3 shows the pseudo code for the data-parallel method whose input includes the number of cores N and the overlapping size “overlapsize”. Suppose the size of input EPF is $m \times n$ (m is the height of the image and n is the image width and we assume $m \times n$), the fringe pattern is equally divided into strips with size $m \times \lfloor n/N \rfloor$ except for the right most strip which has the size of $m \times (n - \lfloor n/N \rfloor (N-1))$, in which $\lfloor \cdot \rfloor$ denotes the floor function. In order to maintain consistency of the cutting edge, the cutting edges of image strips are extended by $m \times overlapsize$, where “overlapsize” is used to eliminate the edge

effects and should be determined by the algorithm to be parallelized. Figure 5.4 shows the division and the synthesis process of this method. For brevity, only one strip is illustrated in Fig. 5.4 to show the division and synthesis process, while all the strips are processed in different cores simultaneously following the same rule. In Fig. 5.4, N is assumed to be four. After image strips are processed by a different worker session, the proposed method copies the corresponding results from all the cores into a buffer to combine the final result in client session. One feature of our proposed approach is that users only need to use a wrapping function to invoke their sequential algorithm instead of modifying it. Thus the codes are very concise.

Figure 5.4 Division and synthesis process for data-parallel approach.

Reprint from Ref. [200] with permission from Elsevier.

5.4 Results and discussions

Figure 5.5 WFF Results using task-parallel and data-parallel approach (a) Original phase map obtained from a holographic system (b) WFF result using task-parallel and data-parallel approach.

Reprint from Ref. [200] with the permission from Elsevier.

Dell Precision T5400 workstation with dual die quad-core 2.5GHz CPUs (equates approximately to 8 physical CPUs) is used to verify our proposed approaches [202]. A 406×568 wrapped phase map shown in Fig. 5.5(a) is used in the experiments. The phase map is reconstructed from a digital holographic system and reflects the thickness distribution of a laser mark on a glass substrate. The parameters “sigmax”=“sigmay”=10, “wxl” = “wyl” = -2, “wxh” = “wyh” = 2 , “wxi”= “wyi”= 0.1 and “thr” = 3 are used in the WFF and WFR algorithm to process the phase map. We choose those parameters because they can be used for many fringe patterns, though for this example, the frequency interval from lower bound to upper bound can be narrower. As a local processing algorithm, the WFF can be also accelerated using the data-parallel approach. Since window size is 81 for WFF and 61 for WFR algorithm, the “overlapsize” for data-parallel approach is 40 for WFF and 30 for WFR. Figure 5.5(b) gives the WFF results by the task-parallel and data-parallel approach, which is exactly the same as that obtained by the

original sequential approach. In the following paragraphs, we will focus on the speedup performances rather than the accuracy because the latter has been thoroughly discussed in previous Chapters.

Figure 5.6 Speedup using proposed parallelism approaches on WFF.

Reprint from Ref. [200] with permission from Elsevier.

As for the speedup performances, the task-parallel approach is applied to convolution-based WFF algorithm shown in Fig. 5.2(b). The speedups from 1.89 to 6.59 are achieved using 2 to 8 cores, which is illustrated in Fig. 5.6. Then the data-parallel approach is applied to the same convolution-based WFF algorithm and the speedup of 1.62 to 3.73 times is achieved with 2 to 8 cores, respectively, which is also shown in Fig. 5.6. Data-parallel approach is slower than that of the task-parallel approach because the data-

parallel approach has more overhead (I.E., extra computation for overlapping size as well as data division and combination processes) than that of the task-parallel approach. Therefore, for embarrassing problem [57] like WFF, task-parallel is preferred for algorithm acceleration using multicore CPU and parallel computing toolbox.

The task and data parallel methods are then applied to the optimized FFT implementation of the WFF algorithm and the results are also shown in Fig. 5.6. It can be seen that, again the task-parallel method is faster than the data-parallel method, as can be expected. Further, it is noted that for both task-parallel and data-parallel based on multicore CPU and parallel computing toolbox, the speedup for the convolution-based WFF algorithm is higher than that for the FFT-based WFF algorithm. This is because convolution-based WFF algorithm is computationally more expensive while the overhead (data division, transmission and combination) for two WFF implementations are almost the same. Thus, the overhead for convolution-based algorithm is less significant and the speedup is more obvious, comparing with the FFT-based implementation.

Though data-parallel approach using multicore CPU and parallel computing toolbox is not as efficient as task-parallel approach for embarrassing problems, it is more general and is applicable for all local processing algorithms. When the data-parallel approach is applied to the WFR algorithm, up to 3.6 times can be achieved for the wrapped phase map shown in Fig. 5.5(a).

5.5 Conclusion

In this Chapter, the WFT-based algorithms were accelerated using MATLAB[®] parallel computing toolbox and multicore CPU. A 406×568 phase map was used for verification. Results showed that using MATLAB[®] parallel computing toolbox and multicore CPU, parallelized WFF could achieve upto 6.59 times speedup using 8 CPU cores. From the experimental results, we can see that there is overhead using multicore CPU for algorithm acceleration, caused by core-level communication and memory management in parallel computing toolbox. When using more computational units in CPU, the overhead cost will be larger than the speedup gain by the parallelism. This impels us to explore using lighter overhead hardware such as GPU. However, because of conciseness of the MATLAB[®] codes, the proposed two methods including task-parallel approach and data-parallel approach can be used to develop prototypes of the WFT based algorithms for other hardware systems. Furthermore, they can be used in other algorithms possessing the same trait as the WFT-based algorithms and the speedup is effortless.

Chapter 6 Real-time 2-D parallel windowed Fourier transform for fringe pattern analysis using Graphic Processing Unit

6.1 Introduction

In previous Chapter, multicore CPU was explored to accelerate the WFT based algorithms. It was shown that the speedup performance was not high due to the limited number of CPU cores and the overhead caused by communications among the CPU cores. Thus we have to seek for a more suitable candidate to accelerate the massive data-parallel problems like the WFT-based algorithms. In Chapter 2, various hardware platforms supporting parallel computing were reviewed for different parallel types of the problems. It was pointed out that GPUs were the most promising candidate to accelerate the data-parallel problems. Thus in this Chapter, we adopt GPUs to accelerate the WFT based algorithms for real-time applications. The main reasons of adopting the GPUs can be summarized as, 1) The WFT based algorithms are data-parallel problems which GPUs are most effective hardware to accelerate. 2) Almost all personal computers can be equipped

with at least one graphics card with a GPU whose cost-performance is higher than the other parallel hardware platforms. 3) By using computer unified device architecture (CUDA) provided by NVIDIA Co. Ltd, only one program is needed mixed with both the codes for CPU and GPU. The underlying compiler can automatically extract and compile the codes for GPU. Thus the codes are concise and easy for management.

A fast parallel WFT-based library is developed in this Chapter using visual studio which can automatically detect the CUDA-enabled GPUs. If no GPU is available in the computer, a sequential version of the WFT based algorithm is called. With NVIDIA GTX295 graphics card, up to $132\times$ speedup compared with the sequential C codes has been achieved. As an example, a fringe pattern with the size 256×256 can be effectively filtered with the speed of 4 fps, which is fast enough for many real-time applications. For the up-to-date GPU models, better acceleration performance can be achieved. In the rest of the Chapter, the parallelization of the WFT algorithms for GPUs and CUDA are discussed. Its application in processing digital holographic fringe patterns is shown. The speedup and accuracy of the parallelized WFT-based algorithms using GPU are discussed. This Chapter is mainly based on our paper “Real-time 2D parallel windowed Fourier transform for fringe pattern analysis using Graphics Processing Unit” which has been published in the journal of “Optics Express” [203].

6.2 Parallelization principles for windowed Fourier transform based algorithms

As introduced by the previous Chapter, the WFF algorithm filters an input fringe pattern through selecting proper frequency bands range from $(\xi, \eta) \in [\xi_l, \xi_h] \times [\eta_l, \eta_h]$, and thresholding the windowed Fourier spectrum $Sf(u, v; \xi, \eta)$ by setting the spectrums with small amplitude to zero; while the WFR algorithm searches the WFT element $h_{\xi, \eta}(x, y) \Big|_{\xi, \eta = (\xi_i, \eta_i)}^{\xi, \eta = (\xi_h, \eta_h)}$ which is the most similar to each local fringe patch. The discrete WFF algorithm can be written as follows,

$$\bar{f}(x, y) = \frac{\xi_i \eta_i}{4\pi^2} \sum_{\eta=\eta_l}^{\eta_h} \sum_{\xi=\xi_l}^{\xi_h} \left[\overline{f(x, y) \otimes h_{\xi, \eta}(x, y)} \otimes h_{\xi, \eta}(x, y) \right]. \quad (6.1)$$

where $h_{\xi, \eta}(x, y) = g(x, y) \exp(j\xi x + j\eta y)$ and $(\xi, \eta) \in [\xi_l, \xi_h] \times [\eta_l, \eta_h]$; the $g(x, y)$ is the window function and usually a Gaussian window expressed in Eq. (3.2) is used; ξ_i and η_i are sampling intervals for ξ and η respectively; $\overline{f(x, y) \otimes h_{\xi, \eta}(x, y)}$ denotes the thresholded spectrum; $\bar{f}(x, y)$ denotes the reconstructed fringe pattern.

It is well known that a convolution can be realized by Fourier transforms. In order to reduce the computation time of Eq. (6.1), Eq. (6.1) can be rewritten as,

$$\bar{f}(x, y) = \frac{\xi_i \eta_i}{4\pi^2} \sum_{\eta=\eta_l}^{\eta_h} \sum_{\xi=\xi_l}^{\xi_h} \mathfrak{T}^{-1} \left(\mathfrak{T} \left\{ \overline{\mathfrak{T}^{-1} [\mathfrak{T}(f) \times \mathfrak{T}(h_{\xi, \eta})]} \right\} \times \mathfrak{T}(h_{\xi, \eta}) \right), \quad (6.2)$$

where $\mathfrak{F}$ and $\mathfrak{F}^{-1}$ denote forward and inverse Fourier transforms, respectively; the coordinates (x, y) on the right hand of the equation are omitted for conciseness. Similarly, the WFR algorithm can be rewritten as the following discrete form,

$$[\omega_x(u, v), \omega_y(u, v)] = \arg \max_{\xi, \eta} \left| \mathfrak{F}^{-1} \left[\mathfrak{F}(f) \times \mathfrak{F}(h_{\xi, \eta}) \right] \right|, \quad (6.3)$$

$$r(u, v) = \left| \mathfrak{F}^{-1} \left[\mathfrak{F}(f) \times \mathfrak{F}(h_{\omega_x(u, v), \omega_y(u, v)}) \right] \right|, \quad (6.4)$$

$$\varphi(u, v) = \text{angle} \left\{ \mathfrak{F}^{-1} \left[\mathfrak{F}(f) \times \mathfrak{F}(h_{\omega_x(u, v), \omega_y(u, v)}) \right] \right\}. \quad (6.5)$$

where $\arg \max_{\xi, \eta}$ denotes the frequencies $(\xi, \eta) \in [\xi_l, \xi_h] \times [\eta_l, \eta_h]$ that make the amplitude of $\mathfrak{F}^{-1} \left[\mathfrak{F}(f) \times \mathfrak{F}(h_{\xi, \eta}) \right]$ achieve maximum value for each pixel (u, v) ; $\omega_x(u, v)$ and $\omega_y(u, v)$ are the estimate frequency map along x and y axis of the WFT spectrum respectively; $r(u, v)$ is the ridges map of the time frequency domain; and $\varphi(u, v)$ is the phase of the retrieved ridges.

Eqs. (6.2-6.5) consist of the following three major types of operations:

- i. The pixel-based operations such as addition, multiplication, division, pixel based comparison (Eq. (6.3)) and thresholding: Since the operations on one pixel are independent of those on other pixels, operations on different pixels can be assigned to different computational units being simultaneous executed. These pixel-based operations are extremely efficient to be executed in a GPU with hundreds of computational units called stream processors.

- ii. The forward and inverse Fourier transforms. By divide-and-conquer paradigm and matrix-vector notation, fast Fourier transform (FFT) has been developed into many parallel versions [204] and can be also efficiently parallelized by the CUDA CUFFT library [205]. It is worth mentioning that, $\mathfrak{F}(f)$ can be pre-computed before the iteration starts; $\mathfrak{F}(h_{\xi,\eta})$ appears twice but is only needed to be computed once for each iteration. Moreover, since a Gaussian window is used, $\mathfrak{F}(h_{\xi,\eta})$ can be theoretically expressed as,

$$\mathfrak{F}(h_{\xi,\eta}) = \sqrt{4\pi\sigma_x\sigma_y} \exp\left\{-\left[\sigma_x^2(u-\xi)^2 + \sigma_y^2(v-\eta)^2\right]/2\right\}, \quad (6.6)$$

which becomes pixel-based operations (I.E., type (i)) with no communication among pixels and can be more efficiently executed in a GPU. Thus, only one forward and two inverse Fourier transforms are computed by CUFFT library in one iteration.

- iii. The iterations for each $(\xi,\eta) \in [\xi_l, \xi_h] \times [\eta_l, \eta_h]$ of the WFT-based algorithms. These iterations are independent from one another so that different iterations can be assigned to different GPUs to be calculated at the same time if multiple GPUs are available in the computer.

6.3 Graphics computing Unit and computer unified device architecture

It can be seen from the above analysis that GPUs are the ideal hardware for accelerating the WFT based algorithms. As implied by its name, GPU was designed to process thousands of threads simultaneously by underlying parallel stream processors for the tasks such as graphics rendering [97]. Among all the hardware accelerators, GPU is outstanding because of its tremendous cost performance, performance per power consumption, and low learning curve. Recently GPGPU (General-Purpose computation in GPUs) was developed to leverage the powerful GPU capability for general computation acceleration, especially for image processing. As introduced in Chapter 2, CUDA is a C-based general-purpose parallel computing programming model, which resolves the programming difficulty and inefficient hardware mapping of traditional graphics APIs for GPGPU. The functions in CUDA codes are called kernels. Based on single-instruction multiple-thread (SIMT) architecture [26], CUDA maps a single kernel to multiple threads to process different input data simultaneously. Under CUDA programming model, GPUs can be viewed as acceleration devices operating as coprocessors to the CPU. The GPUs and the CPU communicate through fast PCI-Express port on the motherboard of a personal computer. Data parallel and computationally intensive functions are offloaded to GPUs to achieve better performance than sequential execution in the CPU. Only one copy of the source code is required with mixed C codes for the CPU and CUDA codes for GPUs. NVIDIA “nvcc” compiler [206] extracts and compiles the codes for GPUs in the source code while the remaining codes are compiled by standard C compiler.

6.4 Parallel WFF and WFR algorithm based on CUDA

As the WFF and the WFR algorithms are typical data-parallel, computationally intensive algorithms that can benefit from the GPU hardware, we provide the parallelizing details and the pseudo code of the parallel WFT based library. As mentioned in previous paragraph, there are mainly three types of operations in the WFT based algorithms. Accordingly, the following three approaches are used to design the fast parallel WFF and WFR algorithm. Firstly, while the pixel-based operations are implemented by loops in a CPU, it can be implemented by a kernel in a GPU with the form “kernel_name<<<grid, block>>> ()”. After the kernel launched from the CPU, it is executed “block×grid” times by “block×grid” different light-weighted CUDA threads in GPU according to SIMT architecture. Through this way, each iteration of pixel-based operations is implemented by a thread and all the threads are equally distributed among the stream processors in the GPU for simultaneous execution [207]. The number of grid and block should be large enough to hide the memory and instruction pipeline latency as well as to be efficiently executed in more advanced GPU models. Thus in our experimental GTX295 graphics card, 256 as block size and 90 as grid size are configured to optimize the parallel WFF and WFR codes.

```

Compute spectrum  $\mathfrak{Z}(f)$  using CUFFT
FOR  $\xi \leftarrow \xi_l$  To  $\xi_h$  do
    FOR  $\eta \leftarrow \eta_l$  To  $\eta_h$  do
        Compute  $\mathfrak{Z}(h_{\xi,\eta})$  according to Eq.(6.7)
        Compute  $\mathfrak{Z}(f) \times \mathfrak{Z}(h_{\xi,\eta})$ 
        Compute  $Sf \leftarrow \mathfrak{Z}^{-1}(\mathfrak{Z}(f) \times \mathfrak{Z}(h_{\xi,\eta}))$  by CUFFT
        Compute  $\overline{Sf}$ 
        Compute  $\mathfrak{Z}(\overline{Sf})$  by CUFFT
        Compute  $\mathfrak{Z}(\overline{Sf}) \times \mathfrak{Z}(h_{\xi,\eta})$ 
        Compute  $g \leftarrow \mathfrak{Z}^{-1}(\mathfrak{Z}(\overline{Sf}) \times \mathfrak{Z}(h_{\xi,\eta}))$  by CUFFT
        Compute  $\overline{f} \leftarrow \overline{f} + g$ 
    ENDFOR
ENDFOR

```

(a)

```

Compute spectrum  $\mathfrak{Z}(f)$  using CUFFT
FOR  $\xi \leftarrow \xi_l$  To  $\xi_h$  do
    FOR  $\eta \leftarrow \eta_l$  To  $\eta_h$  do
        Compute  $\mathfrak{Z}(h_{\xi,\eta})$  according to Eq.(6.7)
        Compute  $\mathfrak{Z}(f) \times \mathfrak{Z}(h_{\xi,\eta})$ 
        Compute  $Sf \leftarrow \mathfrak{Z}^{-1}(\mathfrak{Z}(f) \times \mathfrak{Z}(h_{\xi,\eta}))$  by CUFFT
        IF  $abs(Sf) > r(u, v)$ 
             $r(u, v) \leftarrow abs(Sf)$ 
             $\omega_x(u, v) \leftarrow \xi$ 
             $\omega_y(u, v) \leftarrow \eta$ 
             $\varphi(u, v) \leftarrow angle(Sf)$ 
        ENDIF
    ENDFOR
ENDFOR

```

(b)

Figure 6.1 Pseudo codes of the parallel WFT library (a) The parallel WFF using one GPU (b) The parallel WFR using one GPU. Reprint from Ref. [203] with permission from Optical Society of America.

Secondly, we utilize CUDA parallel FFT library (CUFFT) to accelerate the sequential FFT. The pseudo codes in Figs. 6.1(a) and (b) illustrate the parallel WFF algorithm and WFR algorithm respectively using one GPU. The steps in Fig. 6.1 are launched by CPU and passed on to GPUs for execution. It is worth mentioning that CUFFT library achieves best performance when the transform sizes are powers of a single factor [205]. Hence, we extend the padding size of convolution to the nearest power of a single factor to optimize the parallel algorithm.

Thirdly, for the WFF and WFR algorithms, since the operations for frequencies are independent from one another, the parallel WFT library can automatically detect the available GPUs in a PC and equally divide frequencies bands along x axis by the number of GPUs. In Fig. 6.1, ξ'_l and ξ'_h indicate one of the equally divided frequency bands for each GPU. For Fig. 6.1(a) showing the parallel WFF algorithm, after loading a fringe pattern, each GPU calculates its own section of frequencies independently. Then the CPU reads back the individual result from available GPUs and combines all the results. However, for the parallel WFR algorithm whose pseudo code is shown in Fig. 6.1(b), the calculation results for each frequency have to be compared and depend on one another. When using multiGPU, more complicated synthesis operations rather than simple addition should be conducted on CPU side. Thus using more GPUs would increase the speed performance when the computation load is high, otherwise not.

6.5 Experiments and performance of parallel WFF and WFR using CUDA and GPU

The experimental parallel WFT-based library is tested in quad-core 2.5GHz Intel(R) Xeon(R) CPU E5420 and NVIDIA GTX295 graphics card with 2 GPUs. CUDA 2.3 and visual studio 2005 are adopted as the development software. Then the proposed parallel WFT library based on GPUs are applied to an experimental digital holographic fringe pattern as shown in Fig. 6.2(a). Both single-precision and double-precision versions of the WFT library are implemented. Sequential MATLAB[®] 2008a version (double precision) and “C” version (double precision) of WFT-based algorithms are also tested in the above CPU for comparison. The parameters are set as follows: for the parallel WFF algorithm, $\sigma_x = \sigma_y = 10$, $[\xi, \eta] \in [-1, 1] \times [-1, 1]$, $\xi_i = 0.1$, $\eta_i = 0.1$, and the threshold is 6; and for the parallel WFR, $\sigma_x = \sigma_y = 10$, $[\xi_s, \eta_s] \in [-1, 1] \times [-1, 1]$, $\xi_i = 0.025$ and $\eta_i = 0.025$.

Although the accuracy of the GPU result is slightly sacrificed by using the single-precision arithmetic instead of double-precision, the loss of the accuracy is tolerable. Figure 6.2(a) shows an input fringe pattern in a digital holographic microscopic system. Figures 6.2(b) and (c) show the WFF results by CPU (double-precision) and GPU (single-precision). Their difference is shown in Fig. 6.2(d), which has a peak value of 1.03×10^{-5} radians. If the double-precision arithmetic is adopted for the GPUs, the peak difference decreases to 10^{-15} radians. It is known that the single precision arithmetic is widely used

in real-time applications. Regarding the accuracy, although the double-precision is preferred, single-precision can be adopted for practical use.

Figure 6.2 Sequential and parallel WFF result for processing a digital holographic fringe pattern. (a) a digital holographic fringe pattern, (b) the WFF filtering result using CPU, (c) the WFF filtering result using the proposed system based on GPU, (d) the difference map of the two results. Reprint from Ref. [203] with permission from Optical Society of America.

Table 6.1 Execution time for different size of fringe pattern by WFF and WFR algorithm. Reprint from Ref. [203] with permission from Optical Society of America.

Algorithms for fringe patterns with different sizes	MATLAB [®] sequential (sec)	C sequential (sec)	GTX295 graphics card (double precision) (sec)	GTX295 graphics card (single precision)	
				Execution time (sec)	Speedup against C sequential
WFF(256×256)	30.75	13.77	1.29	0.25	55.08 ×
WFF(512×512)	125.70	54.46	5.11	0.93	58.56 ×
WFF(1024×1024)	459.25	279.14	18.96	3.60	77.54 ×
WFR(256×256)	340.05	149.37	6.54	1.29	115.79 ×
WFR(512×512)	1163.48	606.47	23.16	4.59	132.13 ×
WFR(1024×1024)	5891.58	2424.71	83.01	19.21	126.22 ×

The computation time and the speedup by the proposed parallel WFT-based library are shown in Table 6.1. It can be seen that for the WFF algorithm, GPUs with single-precision give upto 77× and 135× speedup compared with the sequential C codes and sequential MATLAB[®] codes, respectively; while for the WFR algorithm, 132× and 306× speedup can be achieved compared with sequential C codes and MATLAB[®] codes, respectively. Only 0.25 second is needed to process one frame using the parallel WFF to process a 256×256 fringe pattern, which shows its potential in many real-time

measurements. It is worth mentioning that in some applications the frequency bands for WFF and WFR can be narrower and the proposed parallel WFT library based on GPUs can consequently achieve higher frame rate.

6.6 Conclusions and discussions

In this Chapter, parallel windowed Fourier transform based algorithms for fringe pattern analysis using graphics processing unit (GPU) was proposed. The parallelization techniques for GPU were discussed in detail. Real-time WFF was achieved with 4 fps for a 256×256 digital holographic fringe patterns in GTX295 graphics card. The dramatic speedup performance also showed the significance and importance of parallelizing an algorithm. The parallelization approach in GPU is not only technically sound but also practically attractive as a personal computer can simply be equipped with a CUDA supported graphics card through PCI-Express interface in a reasonable price (The most advanced CUDA supported graphics card costs only several hundred US dollars). Finally, the developed parallel WFT library can run seamlessly in all CUDA supported graphics cards after having been compiled in the latest version of CUDA develop kit. With either a more advanced GPU or more available GPUs in a PC, better acceleration performance is expected. Since the WFT based algorithms are highly parallelizable and the computing architecture has come into the era of many-cores, the proposed system can gain more acceptance in real-time applications.

Chapter 7 A real-time pipelined heterogeneous system for WFT-based quality-guided PU algorithms

7.1 Introduction

Our previous work adopted hardware accelerators to improve the speed performance of the WFT based algorithms, which have been introduced in Chapter 5 and 6. It was shown that the GPU was very effective in accelerating the data-parallel problems such as WFT based algorithms while the multicore CPU was more efficient in accelerating the task and pipeline parallel problems. In this Chapter, we combine the previous work to achieve our goal of a high-speed system for phase denoising and unwrapping. Using the pipelined heterogeneous system based on multicore CPU and GPU, the WFF-QG PU algorithm and newly proposed WFRC-QG PU algorithm are accelerated. There are two steps in both the WFF-QG and WFRC-QG algorithms including the denoising step and the unwrapping step. The WFF algorithm and the WFRC algorithm belong to the denoising step by removing the noises on the fringe patterns and providing reliable quality maps for the next step, while the QG step denotes the quality guided phase unwrapping step and unwraps the denoised phase using the quality maps provided by the first step. The second step can

be implemented very fast (in the order of milliseconds) [208] and inefficient to be parallelized comparing with the WFT based algorithms. Thus considering the load balance of the system, it is more efficient for the quality-guided PU algorithm be executed on a sequential computing unit.

In this Chapter, towards real-time fringe analysis, we propose a pipelined heterogeneous system based on multicore CPU and GPU to accelerate the WFF-QG and WFRC-QG PU algorithms. GPU is in charge of highly data-parallel algorithm such as the WFF and WFRC algorithms while multicore CPU executes quality-guided PU algorithm simultaneously but on different fringe frames. These processes are overlapped along the time axis, which is called the pipeline parallelism. Using NVIDIA GTX285 graphics card and quad-core 2.5GHz Intel(R) Xeon(R) CPU E5420, 2.15 fps can be achieved for real-time WFF-QG algorithm with fringes size 256×256 . Moreover, up to $62.3\times$ speedup compared with the sequential C codes in the same CPU has been achieved. In the rest of this Chapter, the WFF-QG and WFRC-QG algorithms are briefly introduced and the high-speed pipelined heterogeneous system based on multicore CPU and GPU is addressed. Their applications in processing simulated wrapped phase maps are demonstrated. Finally, the speedup as well as the accuracy of the proposed system for WFF-QG and WFRC-QG PU algorithms are reported and discussed. The content of this Chapter is mainly based on our previous paper “Real-time pipelined heterogeneous system for windowed Fourier filtering and quality guided phase unwrapping algorithm using Graphic Processing Unit” published in AIP conference proceedings [209].

7.2 Windowed Fourier transform based quality-guided PU algorithm

Since in previous Chapters, the WFT-based algorithms have been thoroughly introduced, here we restrict ourselves in addressing the quality guided PU algorithm. For the result of WFF algorithm $\bar{f}(x,y) = \bar{b}(x,y)\exp[j\bar{\varphi}_w(x,y)]$, $\bar{\varphi}_w(x,y)$ is the filtered wrapped phase, and $\bar{b}(x,y)$ is the filtered amplitude. For the WFRC algorithm, the retrieved filtered phase and ridges map are shown in Eq. (3.8) and (6.4) respectively. They will be used in quality-guided PU algorithm as input wrapped phase and quality map.

The principle of the quality guided phase unwrapping algorithm has been introduced in Chapter 2 and its implementation can be briefly addressed as follows,

- i. Select a starting pixel with high-quality value from the input phase map (first iteration) or the “adjoin list”.
- ii. Unwrap the four neighbors of the selected pixel which have not been unwrapped before and store them in the “adjoin list”.
- iii. Remove the pixel with highest-quality value from the “adjoin list”; unwrap its four neighbors and then placed them in the list.
- iv. Update the “adjoin list” from the pixel with highest quality value to the lowest one. Return to step i. until all the pixels are unwrapped.

It is seen that the quality guided phase unwrapping algorithm is path dependent and inefficient to be parallelized. Furthermore, it can be implemented very fast sequentially in CPU. Thus, it will be executed in one of the cores in CPU.

7.3 Real-time pipelined heterogeneous system based on multicore CPU and GPU

In this subsection, a real-time pipelined heterogeneous system is proposed based on multicore CPU and GPU. It has been shown that GPUs are effective in accelerating data parallel problems while multicore CPU can be used for task and pipeline parallel problems. A heterogeneous system based on multicore CPU and GPU not only sounds good but also practical because of the following reasons:

- 1) The pipelined heterogeneous system can accelerate only the data parallel problems but also the task and pipeline parallel problems.
- 2) Multicore CPU and GPU are readily available in modern personal desktops or laptops.
- 3) The solution is more cost-effective and universal than the FPGA counterparts.

Figure 7.1 Mechanism of real-time pipelined heterogeneous system based on CPU and GPU

Figure 7.2 Pipelined WFF-QG algorithm based on multicore CPU and GPU (a) Stream graph for CPU threads (b) Time sequence for five frames

In general, for a fringe analysis system, input fringe pattern frames are first collected by CCD cameras and processed by various techniques to extract the EPF. Phase unwrapping is then applied to reconstruct continuous phase. In fact, many fringe analysis algorithms are inherently data parallel with pixel-based operations, thus GPU is very useful to leverage the speed performance of those algorithms for real-time applications. Figure 7.1 shows the mechanism of real-time pipelined heterogeneous system based on CPU and GPU. After recorded by CCD camera, fringe pattern frames are transmitted to the workstation or a PC which is equipped with a CUDA enabled graphics card. The whole fringe analysis system can be divided into several load-balanced tasks, each of which is executed by a CPU thread. Data-parallel and computationally intensive functions are offloaded to GPUs to gain better performance than sequential execution in a CPU. GPU communicates with CPU through fast PCI express port on the PC motherboard. As discussed by Chapter 6, the NVIDIA “nvcc” compiler automatically extracts and compiles the codes for GPU while the rest codes are compiled by standard C compiler. A sequence of fringe patterns are assumed to be ready for phase unwrapping using the WFT based quality-guided PU algorithms. In the actual cases, real-time fringe analysis system can process the acquired fringe patterns in a pipelined fashion.

The pipeline mechanism of the tasks in the proposed system is also explained in this subsection. We use GPU to accelerate the WFT based algorithms and adopt multicore CPU for quality-guided PU algorithm. For load balancing, the input and output processes of the system are also executed in the multicore CPU. Shown in Fig. 7.2(a), taken pipelined WFF-QG as example, the frames are first collected by an input thread in CPU; then the communication thread in CPU is used to pass input frames to GPU for processing

and collect the results of the WFF algorithm in GPU; afterwards, the frames are unwrapped by a CPU thread called quality-guided (QG) PU thread; finally, the output thread is used to display the continuous phases in real-time or store them for further analysis. The above four threads in CPU are pipelined along the time axis. In the quad core Xeon(R) E5420 CPU, every thread is allocated an individual CPU core. That is to say, these four CPU threads can perform pipeline parallelism [210]. Figure 7.2(b) indicates the processes for five frames in different time cycle. Suppose we have n frames, execution time for the above four processes in Fig. 7.2(a) are T_i , $T_{c+G_{WFF}}$, T_{qg} and T_o respectively. The total processing time for n frames is,

$$T_{total} = T_a + T_b, \quad (7.1)$$

where

$$T_a = T_i + \max(T_i, T_{c+G_{WFF}}) + \max(T_i, T_{c+G_{WFF}}, T_{qg}) + \max(T_{c+G_{WFF}}, T_{qg}, T_o) + \max(T_{qg}, T_o) + T_o, \quad (7.2)$$

$$T_b = (n-3) * \max(T_i, T_{c+G_{WFF}}, T_{qg}, T_o). \quad (7.3)$$

When n is large enough, the term T_a can be omitted. Hence the total execution time for the proposed fringe analysis system is,

$$T_{total} \approx (n-3) * \max(T_i, T_{c+G_{WFF}}, T_{qg}, T_o). \quad (7.4)$$

By contrast, without the above pipelined architecture, the total processing time for n frames is,

$$T'_{total} = n * (T_i + T_{c+G_{WFF}} + T_{qg} + T_o). \quad (7.5)$$

The frame rate of the real-time pipelined WFF-QG based on CPU and GPU will be ideally $1/\max(T_i, T_{c+G_{WFF}}, T_{qg}, T_o)$ fps instead of $1/(T_i + T_{c+G_{WFF}} + T_{qg} + T_o)$ fps in a sequential

system. It should be noted that even without a quad-core CPU, the proposed pipelined multithread system is more efficient than a single thread program in the CPU. It is because computations in CPU and GPU are asynchronous executed. When GPU processes a fringe pattern frame, CPU threads except for the communication thread can process other frames. Consequently, WFF in GPU and processes in CPU can be executed simultaneously thus yield a higher frame rate than that of a single thread program. The data communication among all CPU threads are buffered by linked lists. The buffer size can be configured according to the memory resources. The scheme of WFRC-QG is the same as that of the WFF-QG algorithm and omitted to make the Chapter concise. It is noted that the time for phase compensation is small and negligible comparing with the WFR algorithm, thus it is simply implemented in the communication thread of the CPU after reading back the WFR results from the GPU.

7.4 Experiments and Performance of the proposed system

The experimental codes are implemented in a program using visual studio 2005 and CUDA 2.3. The compiler automatically loads the GPU threads to the NVIDIA GTX285 graphics card and the CPU threads are executed in quad-core 2.5GHz Intel (R) Xeon E5420 CPU. We implement the proposed pipelined heterogeneous system and provide the sequential version of WFF-QG and WFRC-QG algorithms in the same CPU for comparison. The input wrapped phases are simulated in MATLAB[®] and stored in the hard

disk. Then the system starts by “input thread” reading the wrapped phases frame by frame. The parameters for the WFF algorithm are selected as follows: Gaussian window size is set as $\sigma_x = \sigma_y = 10$, frequency bounds along x and y for integration are $[-1,1] \times [-1,1]$, incremental step for the frequency along x and y axis are both 0.1, and threshold for the WFF algorithm is 6. The parameters for the WFRC algorithm are the same as that of the WFF algorithm except for the incremental step which is 0.025 for the frequency.

In the above experiments, we use single-precision instead of double-precision arithmetic. Although the accuracy of result is slightly sacrificed, the loss of the accuracy is tolerable. The peak difference between single-precision and double-precision arithmetic is around 10^{-5} radians. Regarding the accuracy, although the double-precision is preferred, single-precision is more common for real-time applications. Figure 7.3 shows one frame processed by the WFF-QG in the proposed pipelined system. Figure 7.3(a) is the simulated wrapped phase with Gaussian noise; Figs. 7.3(b) and (c) refer to the phase and amplitude processed by GPU, respectively; Fig. 7.3(d) is the phase error in radians between the WFF-QG result and the simulated phase. The standard deviation of the phase error distribution shown in Fig. 7.3(e) is 0.084 radians.

Figure 7.3 One frame processed by WFF-QG algorithm (a) Simulated wrapped phase with Gaussian noise (b) Filtered wrapped phase by WFF in GPU (c) The quality map provided by WFF for quality-guided PU algorithm (d) Unwrapped phase processed by the WFF-QG algorithm based on the proposed system (e) 3-D phase difference in radians between the proposed system and original simulated phase

In Table 7.1, we show the computation time for the WFF-QG algorithm by sequential program based on CPU only, single thread program based on CPU and GPU, and the pipelined multithreading program based on CPU and GPU, respectively. It shows that by using GPU as a coprocessor, frame rate of WFF-QG algorithm is largely increased by up to 62.3 times. Within the pipelined multithreading architecture, computation times for

other threads are covered by the thread with the largest time. For example, for the fringe pattern with the size of 1024×1024 , WFF algorithm in CPU and GPU is 301.82s and 5.18s respectively, time for the QG phase unwrapping thread is 2.85s, and time for input and output are 1.07s and 4.71s respectively. The difference for the time results of input and output threads is due to reading integers and writing floats. The last line of Table 7.1 can be calculated according to the Eqs. (7.4) and (7.5). Like the WFF-QG algorithm, by using proposed pipelined heterogeneous system based on multicore CPU and GPU, the speedup for the WFRC-QG algorithm to the sequential version can also reach 101 times.

Table 7.1 Frame rate for different size of wrapped phase by WFF-QG algorithm in different system

WFF-QG for wrapped phase with different image sizes	Sequential program based on CPU only (fps)	Single thread program based on CPU+GPU (fps)	Pipelined multithreading program based on CPU+GPU (fps)
256×256	0.046	1.092	2.148
512×512	0.015	0.326	0.833
1024×1024	0.003	0.072	0.187

7.5 Conclusion and discussions

In this Chapter, we combined the work of the previous two Chapters and realized a high-speed pipelined heterogeneous system based on multicore CPU and GPU. High-speed

WFF-QG and WFRC-QG algorithms were obtained for fringe pattern denoising and unwrapping. Since the execution time for the WFF-QG and WFRC-QG algorithms are both independent of the fringe pattern itself, the speed performance is determined for the fringe patterns with a given size. By comparison, the execution times for other popular PU algorithms such as the FLYNN's minimum discontinuity algorithm and the PUMA algorithm also depend on the complexity of the discontinuities on the fringe patterns. Furthermore, the WFF-QG and WFRC-QG algorithm can generate their own quality maps, which have been proved resistant to noises and long phase discontinuities [211]. Nowadays, a PC can simply be equipped with a high-end CUDA supported graphics card which costs only several hundred US dollars. Our proposed pipelined system based on multicore CPU and GPU is undoubtedly a cost-effective solution towards real-time phase denoising and unwrapping. Since CUDA implementations of the WFT based algorithms in GPU are scalable, with more advanced GPU, better acceleration performance is expected. By contrast, the fringe analysis algorithms which are difficult to be parallelized will hardly gain the speedup performance in the modern PC. Finally, a parallel C++ library for the WFF-QG and WFRC-QG algorithms was implemented, which enables the algorithms run seamlessly in all PCs with a multicore CPU and CUDA supported graphics cards.

Chapter 8 Conclusions

8.1 Summary

Despite the considerable development, 2-D phase denoising and unwrapping for fringe pattern analysis remains a tough task. Among the algorithms being more and more complex and time consuming, we seek for a compromise between the robustness and the speed for more applications. Although the WFT-based algorithms cannot deal with the EPFs with under sampling which many Bayesian methods are able to solve [15-21], WFT-based algorithms show great potential in removing the noises and identifying the long discontinuities. Furthermore, the parameters for the WFT-based algorithms are easily set and once the parameters are set the computation time of the WFT-based algorithms is determined for a given size of the fringe patterns. This feature is quite essential for real-time phase measurement because the frame rate will be consistent given a fixed recording device such as a CCD camera. By comparison, the comparable Bayesian methods [17-21] use different energy functions to model the different wrapped phase maps, which involves human intervention to choose suitable energy function and parameters for a given fringe pattern. Thus the WFT-based algorithms are more practical for real-time applications. In this dissertation, a high-speed pipelined heterogeneous system was proposed for phase denoising and unwrapping in fringe pattern analysis. Consisting of multicore CPU and GPU, the system could achieve accurate high-speed WFF-QG and WFRC-QG algorithms.

To achieve an accurate parallel system, we first improved the WFR algorithm by providing two phase compensation methods to correct the phase bias produced by the WFR algorithm. Then a new parametric model based PU algorithm was proposed based on the WFRC algorithm. Experiments and simulations showed that the proposed phase compensation methods could effectively correct the phase bias and improve the MSEs of the WFR algorithm for phase estimation. It was also shown in Chapter 3 that the WFRC algorithm was better than the WFF algorithm for unwrapping some wrapped phase maps.

Secondly, we deduced the theoretical MSEs of the WFRC algorithm in the dissertation for the local frequency and phase estimates to show the high accuracy of the WFRC algorithm. Being a parametric model based algorithm assuming the phase to be a quadratic polynomial, the MSEs of the WFRC algorithm is predictable by statistical analysis. Thus by assuming exponential phase field being contaminated by additive white Gaussian noise, we derived the MSEs of the 1-D WFRC algorithm for local frequency and phase estimates using the first order perturbation technique. Both the MSEs depended on the Gaussian window size σ , second order derivative of the phase c_2 and signal to noise ratio SNR. Then the derived MSEs were verified by Monte Carlo simulations and compared with the standard CRBs. Both theoretical and experimental results showed that although the MSE for frequency estimate was not close to its CRB, the MSE of the phase estimate was. Furthermore, although other parametric model based PU algorithm including the HAF and CPF algorithm were shown closer than the CRBs, the WFRC algorithm was more tolerant to noises when the SNR is small. Thus for true phase

discontinuities, the WFRC algorithm are more applicable than the piecewise HAF method [32] and CPF method [34].

On the other hand, although the WFT-based algorithms are accurate, the high computation load prohibits them from real-time applications. Given the fact that only small speedup gain can be achieved based on fast implementation of the algorithm, using advanced hardware is an alternative method to increase the speed performance of the WFT-based algorithms. During the days when the mainstream computing hardware were based on frequency scaling, software designer did not need to worry about the speed performance of their algorithms because the speed performance would automatically increase when using more advanced computers with higher frequency. The bottleneck of energy consumption for high frequency devices broke their dream, and compelled computing scheme based on frequency scaling being replaced by the one based on multiple computing unit. Nowadays nearly all the modern hardware are based on parallel computing. Thus we have to change our minds and reexamine the those phase unwrapping algorithms for fringe pattern analysis in a parallelizable view. I.E. To be adapted into the new parallel hardware architecture, the former sequential algorithms should have the potential to be parallelized.

Another contribution of our work is to introduce a simple way to parallelize the WFT-based algorithms using MATLAB[®] parallel computing toolbox and multicore CPU. We are also keen to see how the speed performance is using multicore CPU to accelerate the WFT-based algorithms. By using dual die quad-core CPU in Dell Precision T5400 workstation and MATLAB[®] parallel computing toolbox, up to 6.6× speedup was

achieved for accelerating the WFF algorithm and $5.5\times$ speedup for the WFR algorithm with simple modifications of the sequential MATLAB[®] code. From this work, we found that the speedup performance by multicore CPU was small due to the limited number of core in CPU and large communication overhead among the computing units. The proposed parallelization approaches were shown more useful in accelerating heavy load computations. However, using MATLAB[®] and multicore CPU to accelerate the fringe analysis algorithms is simpler with small modifications of the sequential code and easy for codes management. It is even useful in prototyping of the parallelized algorithms for other hardware platforms.

As shown in previous paragraph, the speedup using multicore CPU to accelerate the WFT-based algorithms is far from acceptable in real-time applications. Thus we have to explore more efficient parallel hardware to accelerate the WFT-based algorithms. The hardware platforms based on parallel computing have been reviewed in Chapter 2. It was seen that for the data parallel problems, GPUs were very efficient for the speed performance improvement. On the other hand, the WFT-based algorithms are embarrassing data parallel problems, in which most operations for pixels are independent from one another. Little communication is involved between CPU and GPU in accelerating the WFT-based algorithms. Thus GPUs are the most competent hardware for accelerating WFT-based algorithms. We gave the detailed analysis of the parallelism of the WFT-based algorithms in Chapter 6. Then a parallel WFT-based library was developed for GPUs. By using a GTX295 graphics card, up to $132\times$ and $70\times$ speedup were achieved for single and double precision implementations of the WFR algorithm, respectively. Near real-time WFF algorithm with frame rate of 4 fps was achieved using

the same graphics card. Once the source code of the library is compiled in the latest development environment, the library can be seamlessly executed in all available CUDA supported graphics cards. Despite the tremendous speedup performance, it is more cost-effective than their FPGA counterparts. By using more advanced GPU model, higher speedup performance can be achieved.

At last, based on the above works, we realized high-speed WFF-QG and WFRC-QG algorithm for phase denoising and unwrapping by a pipelined heterogeneous system based on a multicore CPU and GPUs. Having studying the characteristic of the multicore CPU and GPUs, it was seen that multicore CPU is more effective for task and pipeline parallel problems while GPUs are most efficient in accelerating data parallel problems. In the meanwhile, they are easily available in modern personal computer, which means the low cost. It is intuitive to use multicore CPU and GPUs to accelerate a given problem with various parallel types. In the WFF-QG and WFRC-QG algorithms, the WFF and WFRC algorithms are data parallel problems which can be accelerated by GPUs while the quality-guided phase unwrapping is fast and inefficient to be parallelized. Considering the load balance problem, we did not parallelize the quality-guided PU algorithm and adopted a CPU core to execute the algorithm. Towards real-time fringe pattern analysis, the WFT-based algorithms module, the quality-guided PU algorithm module, and other modules such as input and output were executed in pipelined fashion by different cores of the CPU. Thus when the number of fringe frames is large, the frame rates becomes the inverse of the maximum computation time of the above modules. We achieved 2.15 fps for unwrapping 256×256 fringe patterns with noises and long discontinuities using NVIDIA GTX285 graphics card and quad-core 2.5GHz Intel(R) Xeon(R) CPU. The proposed

pipelined architecture contributed about $2\times$ of the total speedup. It should be noted that the development of multicore CPU and GPU is very fast. Thus larger frame rates are expected when using more advanced computer model.

8.2 Future works

In improving the WFT-based algorithm, an adaptive window approach may be used for high frequency changing rates and under sampling issue. Statistical analysis can be explored to give a guaranteed range of the MSEs following the given approach.

For a real-time fringe pattern analysis system, we mainly focus on the most time consuming part such as denoising and the phase unwrapping. Our proposed high-speed system for phase denoising and unwrapping can be integrated into a fringe analysis system in the future by incorporating and parallelizing other steps for fringe analysis. Furthermore, the proposed real-time system can be applied to the fringe patterns in other areas such as InSAR or MRI. A user friendly interface should also be provided to show the real-time physical information as well as the phase change in a period of time. With the development of GPUs, the WFT-based algorithms will not be the bottleneck of the whole fringe analysis system. Other modules should be evaluated and parallelized to balance the computational load for best speedup performance.

The proposed heterogeneous system is expected to be used in wider range of analysis applications such as cell shape analysis in DHM system, 3D phase reconstruction in InSAR and MRI and real-time combustion analysis.

Bibliography

1. P.A. Rosen, "Synthetic aperture radar interferometry," Proc. IEEE **88**, 333-380 (2000).
2. H.D. Griffiths, L.A. Rafik, Z. Meng, C.F.N. Cowan, H. Shafeeu, and D.K. Anthony, "Interferometric synthetic aperture sonar for high-resolution 3-D mapping of the seabed," IEE Proceedings: Radar, Sonar and Navigation **144**, 96-103 (1997).
3. P. Jezzard, and R.S. Balaban, "Correction for geometric distortion in echo planar images from B0 field variations," Magnetic Resonance in Medicine **34**, 65-73 (1995).
4. G.H. Glover, and E. Schneider, "Three-point dixon technique for true water/fat decomposition with B0 inhomogeneity correction," Magnetic Resonance in Medicine **18**, 371-383 (1991).
5. P. Grangeat, *Tomography* (John Wiley & Sons, 2009).
6. K.J. Gåsvik, *Optical metrology* (J. Wiley & Sons, 2002).
7. D.C. Ghiglia, and M.D. Pritt, *Two-dimensional phase unwrapping: theory, algorithms, and software* (Wiley, 1998).
8. L.C. Smith, "Emerging Applications of Interferometric Synthetic Aperture Radar (InSAR) in Geomorphology and Hydrology," Annals of the Association of American Geographers **92**, 385-398 (2002).
9. R.E. Hansen, T.O. Sæbø, K. Gade, and S. Chapman, "Signal processing for AUV based interferometric synthetic aperture sonar," Oceans **5**, 2438-2444 (2003).

10. A. Momose, "Demonstration of phase-contrast X-ray computed tomography using an X-ray interferometer," Nuclear Inst. and Methods in Physics Research, A **352**, 622-628 (1995).
11. E.M. Shapiro, S. Skrtic, K. Sharer, J.M. Hill, C.E. Dunbar, and A.P. Koretsky, "MRI detection of single particles for cellular imaging," Proceedings of the National Academy of Sciences of the United States of America **101**, 10901-10906 (2004).
12. V.R. Singh, O.C. Chee, E. Sim, and A. Asundi, "Characterisation of laser marks using digital holographic microscopy," **7155**, 715509-715509 (2008).
13. J.L. Marroquin, and M. Rivera, "Quadratic regularization functionals for phase unwrapping," J. Opt. Soc. Am. A **12**, 2393-2400 (1995).
14. M. Servin, J.L. Marroquin, D. Malacara, and F.J. Cuevas, "Phase unwrapping with a regularized phase-tracking system," Appl. Opt. **37**, 1917-1923 (1998).
15. L. Guerriero, G. Nico, G. Pasquariello, and S. Stramaglia, "New regularization scheme for phase unwrapping," Appl. Opt. **37**, 3053-3058 (1998).
16. G. Nico, G. Palubinskas, and M. Datcu, "Bayesian approaches to phase unwrapping: Theoretical study," IEEE Transactions on Signal Processing **48**, 2545-2556 (2000).
17. C.W. Chen, and H.A. Zebker, "Two-dimensional phase unwrapping with use of statistical models for cost functions in nonlinear optimization," J. Opt. Soc. Am. A **18**, 338-351 (2001).
18. J.M.B. Dias, and J.M.N. Leitão, "The $\mathbb{Z}\pi M$ algorithm: A method for interferometric image reconstruction in SAR/SAS," IEEE Transactions on Image Processing **11**, 408-422 (2002).

19. J.M. Bioucas-Dias, and G. Valadão, "Phase unwrapping via graph cuts," *IEEE Transactions on Image Processing* **16**, 698-709 (2007).
20. B.D. Jose, V. Katkovnik, J. Astola, and K. Egiazarian, "Absolute phase estimation: adaptive local denoising and global unwrapping," *Appl. Opt.* **47**, 5358-5369 (2008).
21. G. Valadão, and J. Bioucas-Dias, "CAPE: Combinatorial absolute phase estimation," *J. Opt. Soc. Am. A* **26**, 2093-2106 (2009).
22. M. Rivera, and J.L. Marroquin, "Half-quadratic cost functions for phase unwrapping," *Opt. Lett.* **29**, 504-506 (2004).
23. B. Friedlander, and J.M. Francos, "Model based phase unwrapping of 2-D signals," *IEEE Transactions on Signal Processing* **44**, 2999-3007 (1996).
24. Z.P. Liang, "A model-based method for phase unwrapping," *IEEE Transactions on Medical Imaging* **15**, 893-897 (1996).
25. J. Zhong, and J. Weng, "Phase retrieval of optical fringe patterns from the ridge of a wavelet transform," *Opt. Lett.* **30**, 2560-2562 (2005).
26. Z. Wang, and H. Ma, "Advanced continuous wavelet transform algorithm for digital interferogram analysis and processing," *Opt. Eng.* **45**, 045601 (2006).
27. Q. Kemao, W. Gao, and H. Wang, "Windowed Fourier-filtered and quality-guided phase-unwrapping algorithm," *Appl. Opt.* **47**, 5420-5428 (2008).
28. S. Li, W. Chen, and X. Su, "Reliability-guided phase unwrapping in wavelet-transform profilometry," *Appl. Opt.* **47**, 3369-3377 (2008).
29. S.S. Gorthi, and P. Rastogi, "Piecewise polynomial phase approximation approach for the analysis of reconstructed interference fields in digital holographic interferometry," *Journal of Optics A: Pure and Applied Optics* **11**, (2009).

30. S.S. Gorthi, and P. Rastogi, "Analysis of reconstructed interference fields in digital holographic interferometry using the polynomial phase transform," *Measurement Science and Technology* **20**, (2009).
31. S.S. Gorthi, and P. Rastogi, "Numerical analysis of fringe patterns recorded in holographic interferometry using high-order ambiguity function," *Journal of Modern Optics* **56**, 949-954 (2009).
32. S.S. Gorthi, and P. Rastogi, "Windowed high-order ambiguity function method for fringe analysis," *Rev. Sci. Instrum.* **80**, 1-5 (2009).
33. S.S. Gorthi, and P. Rastogi, "Improved high-order ambiguity-function method for the estimation of phase from interferometric fringes," *Opt. Lett.* **34**, 2575-2577 (2009).
34. S.S. Gorthi, G. Rajshekhar, and P. Rastogi, "Strain estimation in digital holographic interferometry using piecewise polynomial phase approximation based method," *Opt. Express* **18**, 560-565 (2010), <http://www.opticsexpress.org/abstract.cfm?URI=oe-18-2-560>.
35. Q. Kemao, W. Gao, and H. Wang, "Windowed Fourier filtered and quality guided phase unwrapping algorithm: on locally high-order polynomial phase," *Appl. Opt.* **49**, 1075-1079 (2010).
36. G. Fornaro, G. Franceschetti, R. Lanari, E. Sansosti, and M. Tesauro, "Global and local phase-unwrapping techniques: a comparison," *J. Opt. Soc. Am. A* **14**, 2702-2708 (1997).
37. J. Strand, and T. Taxt, "Performance Evaluation of Two-Dimensional Phase Unwrapping Algorithms," *Appl. Opt.* **38**, 4333-4344 (1999).

38. A. Baldi, F. Bertolino, and F. Ginesu, "On the performance of some unwrapping algorithms," *Optics and Lasers in Engineering* **37**, 313-330 (2002), <http://www.sciencedirect.com/science/article/pii/S0143816601001233>.
39. Q. Kemao, L.T.H. Nam, L. Feng, and S.H. Soon, "Comparative analysis on some filters for wrapped phase maps," *Appl. Opt.* **46**, 7412-7418 (2007).
40. E. Zappa, and G. Busca, "Comparison of eight unwrapping algorithms applied to Fourier-transform profilometry," *Opt. & Lasers in Eng.* **46**, 106-116 (2008).
41. Q. Kemao, "Two-dimensional windowed Fourier transform for fringe pattern analysis: Principles, applications and implementations," *Opt. & Lasers in Eng.* **45**, 304-317 (2007).
42. A. Patil, and P. Rastogi, "Moving ahead with phase," *Opt. & Lasers in Eng.* **45**, 253-257 (2007).
43. W. Chen, X. Su, Y. Cao, Q. Zhang, and L. Xiang, "Method for eliminating zero spectrum in Fourier transform profilometry," *Opt. & Lasers in Eng.* **43**, 1267-1276 (2005).
44. P. Hlubina, D. Ciprian, J. Luňáček, and R. Chlebus, "Phase retrieval from the spectral interference signal used to measure thickness of SiO₂ thin film on silicon wafer," *Applied Physics B: Lasers and Optics* **88**, 397-403 (2007).
45. M.P. Arroyo, J.A. Bea, N. Andrés, R. Osta, and M. Doblaré, "Force plate for measuring small animal forces by digital speckle pattern interferometry," *Proc. SPIE* **6616**, (2007).
46. P. Cheng, J. Hu, G. Zhang, L. Hou, B. Xu, and X. Wu, "Deformation measurements of dragonfly's wings in free flight by using Windowed Fourier Transform," *Opt. & Lasers in Eng.* **46**, 157-161 (2008).

47. W. Zhao, Y. Chen, L. Shen, and A.Y. Yi, "Refractive index and dispersion variation in precision optical glass molding by computed tomography," *Appl. Opt.* **48**, 3588-3595 (2009).
48. Y. Fu, R.M. Groves, G. Pedrini, and W. Osten, "Kinematic and deformation parameter measurement by spatiotemporal analysis of an interferogram sequence," *Appl. Opt.* **46**, 8645-8655 (2007).
49. L. Huang, Q. Kemao, B. Pan, and A.K. Asundi, "Comparison of Fourier transform, windowed Fourier transform, and wavelet transform methods for phase extraction from a single fringe pattern in fringe projection profilometry," *Opt. & Lasers in Eng.* **48**, 141-148 (2010).
50. Q. Kemao, H. Wang, and W. Gao, "Windowed Fourier transform for fringe pattern analysis: theoretical analyses," *Appl. Opt.* **47**, 5408-5419 (2008).
51. *Real-time, The American Heritage® Dictionary of the English Language, Fourth Edition.* 2003.; Available from: 18 May 2012, <http://www.thefreedictionary.com/real-time>.
52. L. Mertz, "Real-time fringe-pattern analysis," *Appl. Opt.* **22**, 1535-1539 (1983).
53. S. Nakadate, and M. Isshiki, "Real-time fringe pattern processing and its applications," *Proc. SPIE* **2544**, 74-86 (1995).
54. J.-i. Kato, I. Yamaguchi, T. Nakamura, and S. Kuwashima, "Video-rate fringe analyzer based on phase-shifting electronic moiré patterns," *Appl. Opt.* **36**, 8403-8412 (1997).
55. C.R. Coggrave, and J.M. Huntley, "High-speed surface profilometer based on a spatial light modulator and pipeline image processor," *Opt. Eng.* **38**, 1573-1581 (1999).

56. M.A. Gdeisat, D.R. Burton, and M.J. Lalor, "Real-time hybrid fringe pattern analysis using a linear digital phase locked loop for demodulation and unwrapping," *Measurement Science and Technology* **11**, 1480-1492 (2000).
57. D.A. Patterson, and J.L. Hennessy, *Computer organization and design: the hardware/software interface* (Elsevier Morgan Kaufmann, 2009).
58. C. Quan, H. Niu, and C.J. Tay, "An improved windowed Fourier transform for fringe demodulation," *Opt. & Laser Tech.* **42**, 126-131 (2010).
59. Mathworks. *Parallel computing toolbox*. 2009; Available from: <http://www.mathworks.com/products/parallel-computing>.
60. GPUORG. *GPGPU*. 2011; Available from: <http://gpgpu.org/>
61. R.M. Goldstein, H.A. Zebker, and C.L. Werner, "Satellite radar interferometry: Two-dimensional phase unwrapping," *Radio Sci.* **23**, 713-720 (1988).
62. X. Wei, and I. Cumming, "A region growing algorithm for InSAR phase unwrapping," *Geoscience and Remote Sensing Symposium*, 1996. IGARSS '96. 'Remote Sensing for a Sustainable Future.', International **4**, 2044-2046 vol.2044 (1996).
63. C. Prati, M. Giani, and N. Leuratti, "SAR Interferometry: A 2-D Phase Unwrapping Technique Based On Phase And Absolute Values Informations," *Geoscience and Remote Sensing Symposium*, 1990. IGARSS '90. 'Remote Sensing Science for the Nineties'. 10th Annual International, 2043-2046 (1990).
64. T.J. Flynn, "Consistent 2-D phase unwrapping guided by a quality map," *Geoscience and Remote Sensing Symposium*, 1996. IGARSS '96. 'Remote Sensing for a Sustainable Future.', **4**, 2057-2059 (1996).

65. T.J. Flynn, "Two-dimensional phase unwrapping with minimum weighted discontinuity," J. Opt. Soc. Am. A **14**, 2692-2701 (1997), <http://josaa.osa.org/abstract.cfm?URI=josaa-14-10-2692>.
66. C.W. Chen, and H.A. Zebker, "Network approaches to two-dimensional phase unwrapping: Intractability and two new algorithms," J. Opt. Soc. Am. A **17**, 401-414 (2000).
67. T. Mario Costantini, "A novel phase unwrapping method based on network programming," IEEE Transactions on Geoscience and Remote Sensing **36**, 813-821 (1998).
68. M. Zhao, L. Huang, Q. Zhang, X. Su, A. Asundi, and Q. Kemao, "Quality-guided phase unwrapping technique: comparison of quality maps and guiding strategies," Appl. Opt. **50**, 6214-6224 (2011).
69. P. O'Shea, "A fast algorithm for estimating the parameters of a quadratic FM signal," IEEE Transactions on Signal Processing **52**, 385-393 (2004).
70. J. Zhong, and J. Weng, "Spatial Carrier-Fringe Pattern Analysis by Means of Wavelet Transform: Wavelet Transform Profilometry," Appl. Opt. **43**, 4993-4998 (2004).
71. S.G. Mallat, *A wavelet tour of signal processing* (Academic Press, 1999).
72. A.V. Oppenheim, and R.W. Schaffer, *Discrete-time signal processing* (Prentice Hall, 2010).
73. H. Cramér, *Mathematical methods of statistics* (Princeton University Press, 1999).
74. C.R. Rao, "Information and the accuracy attainable in the estimation of statistical parameters," Bull. Calcutta Math. Soc **37**, 8189 (1946).

75. A.C. Tamhane, and D.D. Dunlop, *Statistics and data analysis: from elementary to intermediate* (Prentice Hall, 2000).
76. S. Peleg, and B. Porat, "Linear FM signal parameter estimation from discrete-time observations," *IEEE Transactions on Aerospace and Electronic Systems* **27**, 607-616 (1991).
77. W. Gao, and Q. Kemao, "Parallel computing in experimental mechanics and optical measurement: A review," *Optics and Lasers in Engineering* **50**, 608-617 (2012).
78. V. Kumar, *Introduction to parallel computing: design and analysis of algorithms* (Benjamin/Cummings Pub. Co., 1994).
79. G.R. Perrin, and A. Darte, *The data parallel programming model: foundations, HPF realization, and scientific applications* (Springer, 1996).
80. M.I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 151-162 (2006).
81. G.M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, (ACM, Atlantic City, New Jersey, 1967), 483-485.
82. M.J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers* **C-21**, 948-960 (1972).
83. R. Duncan, "A survey of parallel computer architectures," *Computer* **23**, 5-16 (1990).
84. B. Wilkinson, *Grid computing: techniques and applications* (CRC Press, 2009).

85. N. Antonopoulos, and L. Gillam, *Cloud Computing: Principles, Systems and Applications* (Springer, 2010).
86. J. Gustafson, "The program of grand challenge problems: expectations and results," *Parallel Algorithms/Architecture Synthesis*, 1997. Proceedings. Second Aizu International Symposium, 2-7 (1997).
87. D.E. Culler, J.P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach* (Morgan Kaufmann Publishers, 1999).
88. G. Blake, R.G. Dreslinski, and T. Mudge, "A survey of multicore processors: A review of their common attributes," *IEEE Signal Processing Magazine* **26**, 26-37 (2009).
89. IBM. *High performance computing & multicore architecture*. 2011; Available from: http://www-07.ibm.com/in/ur/pdf/High_Performance_Computing.pdf.
90. Intel. *Corei7*. 2011; Available from: <http://www.intel.com/products/processor/corei7ee/index.htm>.
91. H. Kim, and R. Bond, "Multicore software technologies: A survey," *IEEE Signal Processing Magazine* **26**, 80 (2009).
92. S.W. Keckler, K. Olukotun, and H.P. Hofstee, *Multicore Processors and Systems* (Springer, 2009).
93. T.D. Williams, "Image understanding tools," *Proc. IEEE, International Conference on Pattern Recognition* **2**, 606-610 (1990).
94. V. Kurz, "Vector and parallel processors in computational science III," *Parallel Computing* **6**, 127-129 (1988).
95. R.M. Hord, *Parallel supercomputing in SIMD architectures* (CRC Press, 1990).

96. N. Kehtarnavaz, and M.N. Gamadia, *Real-time image and video processing: from research to reality* (Morgan & Claypool Publishers, 2006).
97. R. Scott, "Stream processor architecture," Springer International Series in Engineering and Computer Science **664**, (2001).
98. NVIDIA. *CUDA*. 2011; Available from:
http://www.nvidia.com/object/what_is_cuda_new.html.
99. Top500ORG. *Top 500 supercomputers*. 2011; Available from:
<http://www.top500.org/>.
100. J. Gummaraju, L. Morichetti, M. Houston, B. Sander, B.R. Gaster, and B. Zheng, "Twin peaks: A software platform for heterogeneous computing on general-purpose and graphics processors," Proceedings of the 19th international conference on Parallel architectures and compilation techniques, 205-215 (2010).
101. T. Chen, R. Raghavan, J.N. Dale, and E. Iwata, "Cell broadband engine architecture and its first implementation - A performance view," IBM Journal of Research and Development **51**, 559-572 (2007).
102. SGI. *Onyx*. 2011; Available from: <http://www.sgi.com/products/>.
103. R. Hossain, *High performance ASIC design: using synthesizable domino logic in an ASIC flow* (Cambridge University Press, 2008).
104. A. Jerraya, H. Tenhunen, and W. Wolf, "Guest Editors' Introduction: Multiprocessor Systems-on-Chips," Computer **38**, 36-40 (2005).
105. J.J. Rodriguez-Andina, M.J. Moure, and M.D. Valdes, "Features, design tools, and application domains of FPGAs," IEEE Transactions on Industrial Electronics **54**, 1810-1823 (2007).

106. S. Donthi, and R.L. Haggard, "A survey of dynamically reconfigurable FPGA devices," Proceedings of the 35th Southeastern Symposium on System Theory, 422-426 (2003).
107. M. Lehky, and S. Bilik, "Reducing FPGA design modification time," Proc. of VIUF Lehky, 143-149 (1997).
108. K.S. Yeo, K.T. Ng, and Z.H. Kong, *Intellectual Property for Integrated Circuits* (J Ross Pub, 2010).
109. J. Nurmi, *Processor design: system-on-chip computing for ASICs and FPGAs* (Springer, 2007).
110. Mentor. *HandleC*. 2011; Available from:
<http://www.mentor.com/products/fpga/handel-c/>.
111. Playstation. *Playstation 3*. 2011; Available from: <http://us.playstation.com/ps3/>, .
112. TexasInstruments. *TMS320C6678 DSP*. 2011; Available from:
<http://focus.ti.com/docs/prod/folders/print/tms320c6678.html>.
113. Xilinx. *Virtex-6 SX475T*. 2011; Available from:
<http://www.xilinx.com/products/virtex6/sxt.htm>.
114. NVIDIA. *NVIDIA® GeForce® GTX580*. 2011; Available from:
<http://www.nvidia.com/object/product-geforce-gtx-580-us.html>.
115. OpenmpORG. *OpenMP*. 2011; Available from: <http://openmp.org/wp/>.
116. SourcewareORG. *PThreads*. 2011; Available from:
<http://sourceware.org/pthreads-win32/>.
117. KhronosORG. *OpenCL*. 2011; Available from: <http://www.khronos.org/opencv/>.
118. TexasInstruments. *MCSDK*. 2011; Available from:
<http://focus.ti.com/docs/toolsw/folders/print/bioslinuxmcsdk.html>.

119. Y. Pu, and H. Meng, "Four-dimensional dynamic flow measurement by holographic particle image velocimetry," *Appl. Opt.* **44**, 7697-7708 (2005).
120. S. Satake, T. Anraku, H. Kanamori, T. Kunugi, K. Sato, and T. Ito, "Study on high speed parallel algorithm using PC grid environment for visualization measurements by Digital Holographic Particle Tracking Velocimetry," *Computer Physics Communications* **178**, 1-7 (2008).
121. J.P. Fugal, T.J. Schulz, and R.A. Shaw, "Practical methods for automated reconstruction and characterization of particles in digital in-line holograms," *Measurement Science and Technology* **20**, (2009).
122. A.J. Page, L. Ahrenberg, and T.J. Naughton, "Low memory distributed reconstruction of large digital holograms," *Opt. Express* **16**, 1990-1995 (2008), <http://www.opticsexpress.org/abstract.cfm?URI=oe-16-3-1990>.
123. T.W. Ng, K.T. Ang, and G. Argentini, "Temporal fringe pattern analysis with parallel computing," *Appl. Opt.* **44**, 7125-7129 (2005).
124. S.-i. Satake, H. Kanamori, T. Kunugi, K. Sato, T. Ito, and K. Yamamoto, "Parallel computing of a digital hologram and particle searching for microdigital-holographic particle-tracking velocimetry," *Appl. Opt.* **46**, 538-543 (2007).
125. J. Franco, G. Bernabé, J. Fernández, and M. Ujaldón, "Parallel 3D fast wavelet transform on manycore GPUs and multicore CPUs," *Procedia Computer Science* **1**, 1101-1110 (2010).
126. S. Zhang, and P.S. Huang, "High-resolution, real-time three-dimensional shape measurement," *Opt. Eng.* **45**, 123601 (2006).
127. B. Ionția, P.C. Logofătu, and D. Apostol, "Precise interferometric length measurement using real-time fringe fitting," *Optik* **122**, 868-871 (2011).

128. A.J. Moore, and J.R. Tyrer, "Two-dimensional strain measurement with ESPI," *Opt. & Lasers in Eng.* **24**, 381-402 (1996).
129. C.R. Coggrave, and J.M. Huntley, "Real-time visualisation of deformation fields using speckle interferometry and temporal phase unwrapping," *Opt. & Lasers in Eng.* **41**, 601-620 (2004).
130. M. Lucente, "Interactive computation of holograms using a look-up table," *Journal of Electronic Imaging* **2**, 28-34 (1993).
131. C. Petz, and M. Magnor, "Fast hologram synthesis for 3D geometry models using graphics hardware," *Proc. SPIE* **5005**, 266-275 (2003).
132. V.M. Bove Jr, W.J. Plesniak, T. Quentmeyer, and J. Barabas, "Real-time holographic video images with commodity PC hardware," *Proc. SPIE* **5664**, 255-262 (2005).
133. W. Plesniak, M. Halle, V.M. Bove Jr, J. Barabas, and R. Pappu, "Reconfigurable image projection holograms," *Opt. Eng.* **45**, 115801 (2006).
134. N. Masuda, T. Ito, T. Tanaka, A. Shiraki, and T. Sugie, "Computer generated holography using a graphics processing unit," *Opt. Express* **14**, 603-608 (2006), <http://www.opticsexpress.org/abstract.cfm?URI=oe-14-2-603>.
135. L. Ahrenberg, P. Benzie, M. Magnor, and J. Watson, "Computer generated holography using parallel commodity graphics hardware," *Opt. Express* **14**, 7636-7641 (2006), <http://www.opticsexpress.org/abstract.cfm?URI=oe-14-17-7636>.
136. M. Janda, I. Hanák, and L. Onural, "Hologram synthesis for photorealistic reconstruction," *J. Opt. Soc. Am. A* **25**, 3083-3096 (2008).

137. H. Kang, T. Yamaguchi, H. Yoshikawa, S.-C. Kim, and E.-S. Kim, "Acceleration method of computing a compensated phase-added stereogram on a graphic processing unit," *Appl. Opt.* **47**, 5784-5789 (2008).
138. H. Kang, F. Yara, and L. Onural, "Graphics processing unit accelerated computation of digital holograms," *Appl. Opt.* **48**, H137-H143 (2009).
139. Y. Pan, X. Xu, S. Solanki, X. Liang, R.B.A. Tanjung, C. Tan, and T.-C. Chong, "Fast CGH computation using S-LUT on GPU," *Opt. Express* **17**, 18543-18555 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-21-18543>.
140. R.H.Y. Chen, and T.D. Wilkinson, "Computer generated hologram with geometric occlusion using GPU-accelerated depth buffer rasterization for three-dimensional display," *Appl. Opt.* **48**, 4246-4255 (2009).
141. R. H.-Y. Chen, and T.D. Wilkinson, "Computer generated hologram from point cloud using graphics processor," *Appl. Opt.* **48**, 6841-6850 (2009).
142. T. Shimobaba, H. Nakayama, N. Masuda, and T. Ito, "Rapid calculation algorithm of Fresnel computer-generated-hologram using look-up table and wavefront-recording plane methods for three-dimensional display," *Opt. Express* **18**, 19504-19509 (2010), <http://www.opticsexpress.org/abstract.cfm?URI=oe-18-19-19504>.
143. A. Shiraki, N. Takada, M. Niwa, Y. Ichihashi, T. Shimobaba, N. Masuda, and T. Ito, "Simplified electroholographic color reconstruction system using graphics processing unit and liquid crystal display projector," *Opt. Express* **17**, 16038-16045 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-18-16038>.
144. H. Nakayama, N. Takada, Y. Ichihashi, S. Awazu, T. Shimobaba, N. Masuda, and T. Ito, "Real-time color electroholography using multiple graphics processing

- units and multiple high-definition liquid-crystal display panels," *Appl. Opt.* **49**, 5993-5996 (2010).
145. J. Carpenter, and T.D. Wilkinson, "Graphics processing unit-accelerated holography by simulated annealing," *Opt. Eng.* **49**, 095801 (2011).
146. T. Shimobaba, T. Ito, N. Masuda, Y. Ichihashi, and N. Takada, "Fast calculation of computer-generated-hologram on AMD HD5000 series GPU and OpenCL," *Opt. Express* **18**, 9955-9960 (2010), <http://www.opticsexpress.org/abstract.cfm?URI=oe-18-10-9955>.
147. M. Reicherter, S. Zwick, T. Haist, C. Kohler, H. Tiziani, and W. Osten, "Fast digital hologram generation and adaptive force measurement in liquid-crystal-display-based holographic tweezers," *Appl. Opt.* **45**, 888-896 (2006).
148. T. Haist, M. Reicherter, M. Wu, and L. Seifert, "Using graphics boards to compute holograms," *Computing in Science and Engineering* **8**, 8-13 (2006).
149. T. Shimobaba, Y. Sato, J. Miura, M. Takenouchi, and T. Ito, "Real-time digital holographic microscopy using the graphic processing unit," *Opt. Express* **16**, 11776-11781 (2008), <http://www.opticsexpress.org/abstract.cfm?URI=oe-16-16-11776>.
150. L. Wang, J. Zhao, J. Di, and H. Jiang, "Fast extended focused imaging in digital holography using a graphics processing unit," *Opt. Lett.* **36**, 1620-1622 (2011).
151. S. Zhang, D. Royer, and S.-T. Yau, "GPU-assisted high-resolution, real-time 3-D shape measurement," *Opt. Express* **14**, 9120-9129 (2006), <http://www.opticsexpress.org/abstract.cfm?URI=oe-14-20-9120>.

152. S. Liu, P. Li, and Q. Luo, "Fast blood flow visualization of high-resolution laser speckle imaging data using graphics processing unit," *Opt. Express* **16**, 14321-14329 (2008), <http://www.opticsexpress.org/abstract.cfm?URI=oe-16-19-14321>.
153. A. Espinosa-Romero, and R. Legarda-Saenz, "GPU based real-time quadrature transform method for 3-D surface measurement and visualization," *Opt. Express* **19**, 12125-12130 (2011), <http://www.opticsexpress.org/abstract.cfm?URI=oe-19-13-12125>.
154. F.C. Cheong, B. Sun, R. Dreyfus, J. Amato-Grill, K. Xiao, L. Dixon, and D.G. Grier, "Flow visualization and flow cytometry with holographic video microscopy," *Opt. Express* **17**, 13071-13079 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-15-13071>.
155. M. Lucente, and T.A. Galyean, "Rendering interactive holographic images," *Computer Graphics Proc., Annual Conf. Series. ACM SIGGRAPH*, 387-394 (1995).
156. A. Ritter, J. Böttger, O. Deussen, M. König, and T. Strothotte, "Hardware-based Rendering of Full-parallax Synthetic Holograms," *Appl. Opt.* **38**, 1364-1369 (1999).
157. M.W. Halle, and A.B. Kropp, "Fast computer graphics rendering for full parallax spatial displays," *Proc. SPIE* **3011**, 105-112 (1997).
158. D.R. Burton, A.J. Goodall, J.T. Atkinson, and M.J. Lalor, "The use of carrier frequency shifting for the elimination of phase discontinuities in Fourier transform profilometry," *Opt. & Lasers in Eng.* **23**, 245-257 (1995).
159. A.F. Doval, J.L. Fernández, M. Pérez-Amor, J.D.R. Valera, and J.D.C. Jones, "Contrast enhanced and phase controlled stroboscopic additive fibre optic TV-

- holography for whole field out-of-plane vibration analysis," *Opt. & Lasers in Eng.* **25**, 323-342 (1996).
160. A.P. Lee, C.F. McConaghy, G. Sommargren, P. Krulevitch, and E.W. Campbell, "Vertical-actuated electrostatic comb drive with in situ capacitive position correction for application in phase shifting diffraction interferometry," *Journal of Microelectromechanical Systems* **12**, 960-971 (2003).
161. M.V. Aguanno, F. Lakestani, M.P. Whelan, and M.J. Connelly, "Heterodyne speckle interferometer for full-field velocity profile measurements of a vibrating membrane by electronic scanning," *Opt. & Lasers in Eng.* **45**, 677-683 (2007).
162. M.J. Connelly, P.M. Szecówka, R. Jallapuram, S. Martin, V. Toal, and M.P. Whelan, "Multipoint laser Doppler vibrometry using holographic optical elements and a CMOS digital camera," *Opt. Lett.* **33**, 330-332 (2008).
163. C. Zhao, and J.H. Burge, "Vibration-Compensated Interferometer for Surface Metrology," *Appl. Opt.* **40**, 6215-6222 (2001).
164. M.A. Zumberge, J. Berger, M.A. Dzieciuch, and R.L. Parker, "Resolving Quadrature Fringes in Real Time," *Appl. Opt.* **43**, 771-775 (2004).
165. T.J. Park, H.S. Choi, C.S. Han, and Y.W. Lee, "Real-time precision displacement measurement interferometer using the robust discrete time Kalman filter," *Opt. & Laser Tech.* **37**, 229-234 (2005).
166. K.-Y. Huang, and C.-H. Yeh, "Optical measurement system for error motions of small shafts," *Opt. Eng.* **47**, 053601-053610 (2008).
167. K. Sasagawa, A. Kanno, and M. Tsuchiya, "Real-time digital signal processing for live electro-optic imaging," *Opt. Express* **17**, 15641-15651 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-18-15641>.

168. R.G. Racca, and J.M. Dewey, "High speed time-resolved holographic interferometer using solid-state shutters," *Opt. & Laser Tech.* **22**, 199-204 (1990).
169. L. Yue-Guang, Z. Xia, C. Maosheng, and Y. Yunchou, "Fast image-processing technique in phase-shifting holographic interferometry," *Opt. & Laser Tech.* **25**, 163-166 (1993).
170. T. Ito, H. Eldeib, K. Yoshida, S. Takahashi, T. Yabe, and T. Kunugi, "Special-purpose computer for holography HORN-2," *Computer Physics Communications* **93**, 13-20 (1996).
171. T. Shimobaba, N. Masuda, T. Sugie, S. Hosono, S. Tsukui, and T. Ito, "Special-purpose computer for holography HORN-3 with PLD technology," *Computer Physics Communications* **130**, 75-82 (2000).
172. T. Ito, N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, and T. Sugie, "Special-purpose computer HORN-5 for a real-time electroholography," *Opt. Express* **13**, 1923-1932 (2005), <http://www.opticsexpress.org/abstract.cfm?URI=oe-13-6-1923>.
173. N. Masuda, T. Ito, K. Kayama, H. Kono, S.-i. Satake, T. Kunugi, and K. Sato, "Special purpose computer for digital holographic particle tracking velocimetry," *Opt. Express* **14**, 587-592 (2006), <http://www.opticsexpress.org/abstract.cfm?URI=oe-14-2-587>.
174. Y. Abe, N. Masuda, H. Wakabayashi, Y. Kazo, T. Ito, S.-i. Satake, T. Kunugi, and K. Sato, "Special purpose computer system for flow visualization using holography technology," *Opt. Express* **16**, 7686-7692 (2008), <http://www.opticsexpress.org/abstract.cfm?URI=oe-16-11-7686>.

175. N. Masuda, T. Sugie, T. Ito, S. Tanaka, Y. Hamada, S.I. Satake, T. Kunugi, and K. Sato, "Special purpose computer system with highly parallel pipelines for flow visualization using holography technology," *Computer Physics Communications* **181**, 1986-1989 (2010).
176. Y. Ichihashi, H. Nakayama, T. Ito, N. Masuda, T. Shimobaba, A. Shiraki, and T. Sugie, "HORN-6 special-purpose clustered computing system for electroholography," *Opt. Express* **17**, 13895-13903 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-16-13895>.
177. T. Shimobaba, and T. Ito, "An efficient computational method suitable for hardware of computer-generated hologram with phase computation by addition," *Computer Physics Communications* **138**, 44-52 (2001).
178. T. Ito, and T. Shimobaba, "One-unit system for electroholography by use of a special-purpose computational chip with a high-resolution liquid-crystal display toward a three-dimensional television," *Opt. Express* **12**, 1788-1793 (2004), <http://www.opticsexpress.org/abstract.cfm?URI=oe-12-9-1788>.
179. Y. Ichihashi, N. Masuda, M. Tsuge, H. Nakayama, A. Shiraki, T. Shimobaba, and T. Ito, "One-unit system to reconstruct a 3-D movie at a video-rate via electroholography," *Opt. Express* **17**, 19691-19697 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-22-19691>.
180. T. Lenart, M. Gustafsson, and V. Öwall, "A hardware acceleration platform for digital holographic imaging," *Journal of Signal Processing Systems* **52**, 297-311 (2008).
181. R. Veitch, D.C. Hendry, and J. Watson, "Reconfigurable hardware applied to holographic reconstruction," *OCEANS 2007*, 1-6 (2007).

182. I. Hanak, P. Zemcik, M. Zadnik, and A. Herout, "Hologram synthesis accelerated in field programmable gate array by partial quadratic interpolation," *Opt. Eng.* **48**, 085802-085807 (2009).
183. R. Höfling, and E. Ahl, "ALP: Universal DMD controller for metrology and testing," *Proc. SPIE* **5289**, 322-329 (2004).
184. S. Ri, M. Fujigaki, and Y. Morimoto, "Single-shot three-dimensional shape measurement method using a digital micromirror device camera by fringe projection," *Opt. Eng.* **48**, 103605-103608 (2009).
185. Z. Gao, and C. Xiong, "High-throughput implementation of lifting-based discrete wavelet transforms using look-ahead pipelining," *Opt. Eng.* **49**, 107003-107007 (2010).
186. M. Fujigaki, Y. Sasatani, A. Masaya, H. Kondo, M. Nakabo, T. Hara, Y. Morimoto, D. Asai, T. Miyagi, and N. Kurokawa, "Development of sampling Moire camera for real-time phase analysis," *Applied Mechanics and Materials* **83**, 48-53 (2011).
187. E. Buckley, "Real-time error diffusion for signal-to-noise ratio improvement in a holographic projection system," *IEEE/OSA Journal of Display Technology* **7**, 70-76 (2011).
188. J. Ni, X. Li, T. He, and G. Wang, "Review of parallel computing techniques for computed tomography image reconstruction," *Current Medical Imaging Reviews* **2**, 405-414 (2006).
189. J.W. Goodman, *Speckle Phenomena in Optics: Theory and Applications* (Roberts & Company, 2007).

190. J.W. Goodman, *Speckle phenomena in optics : theory and applications* (Roberts & Co., Englewood (Colorado), 2007).
191. M. Takeda, and K. Mutoh, "Fourier transform profilometry for the automatic measurement of 3-D object shapes," *Appl. Opt.* **22**, 3977-3982 (1983), <http://ao.osa.org/abstract.cfm?URI=ao-22-24-3977>.
192. V. Srinivasan, H.C. Liu, and M. Halioua, "Automated phase-measuring profilometry of 3-D diffuse objects," *Appl. Opt.* **23**, 3105-3108 (1984), <http://ao.osa.org/abstract.cfm?URI=ao-23-18-3105>.
193. Z. Wang, D.A. Nguyen, and J.C. Barnes, "Some practical considerations in fringe projection profilometry," *Optics and Lasers in Engineering* **48**, 218-225 (2010).
194. S. Peleg, B. Porat, and B. Friedlander, "Achievable accuracy in estimating the instantaneous phase and frequency of a constant amplitude signal," *IEEE Transactions on Signal Processing* **41**, 2216-2224 (1993).
195. W. Gao, and Q. Kemao, "Statistical analysis for windowed Fourier ridge algorithm in fringe pattern analysis," *Appl. Opt.* **51**, 328-337 (2012).
196. W. Gao, and Q. Kemao, "Statistical analysis of 2D windowed Fourier ridges algorithm," in *5th International Conference on Optical Measurement Techniques for Structures & Systems*, (4 & 5 April 2012 - Antwerp, Belgium, 2012).
197. Intel. *Teraflops Research Chip*. 2011; Available from: <http://techresearch.intel.com/articles/Tera-Scale/1449.htm>.
198. Mathworks. *MATLAB®*. 2011; Available from: <http://www.mathworks.com/>.
199. Mathworks. *Reduction variable*. Retrieved by Jun, 2012; Available from: <http://www.mathworks.com/help/toolbox/distcomp/brdqtjj-1.html>.

200. 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," *Opt. & Lasers in Eng.* **47**, 1286-1292 (2009).
201. Mathworks. *Parallel Computing Toolbox™ 4 User's Guide*. 2009; Available from: <http://www.mathworks.com/>.
202. Intel. *Intel® Xeon®*. 2009; Available from: <http://ark.intel.com/cpu.aspx?groupId=33927>.
203. W. Gao, N.T.T. Huyen, H.S. Loi, and Q. Kemao, "Real-time 2D parallel windowed Fourier transform for fringe pattern analysis using Graphics Processing Unit," *Opt. Express* **17**, 23147-23152 (2009), <http://www.opticsexpress.org/abstract.cfm?URI=oe-17-25-23147>.
204. C.F.V. Loan, S.f. Industrial, and A. Mathematics, *Computational frameworks for the fast fourier transform* (SIAM, 1992).
205. NVIDIA. *CUDA CUFFT library 2.3*. 2009; Available from: http://www.nvidia.com/object/cuda_develop.html.
206. NVIDIA. *The CUDA compiler Driver NVCC*. 2009; Available from: http://moss.csc.ncsu.edu/~mueller/cluster/nvidia/2.0/nvcc_2.0.pdf.
207. NVIDIA. *CUDA Programming Guide Version 2.3*. 2009; Available from: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/
208. M. Zhao, L. Huang, Q. Zhang, Q. Kemao, X. Su, and A. Asundi, "Quality-guided phase unwrapping technique: comparison of quality maps and guiding strategies," *Appl. Opt.* **50**, 6214-6224 (2011).

- 209. W. Gao, and Q. Kemao, "Real-time pipelined heterogeneous system for windowed Fourier filtering and quality guided phase unwrapping algorithm using Graphic Processing Unit," AIP Conference Proc. **1236**, 129-134 (2010).
- 210. W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A practical approach to exploiting coarse-grained pipeline parallelism in C programs," Proceedings of the Annual International Symposium on Microarchitecture, 356-368 (2007).
- 211. Q. Kemao, H. Wang, and W. Gao, "Some recent developments of windowed fourier transform for fringe pattern analysis," AIP Conference Proc. **1236**, 106-111 (2010).